BỘ LAO ĐỘNG - THƯƠNG BINH VÀ XÃ HỘI

TỔNG CỤC DẠY NGHỀ

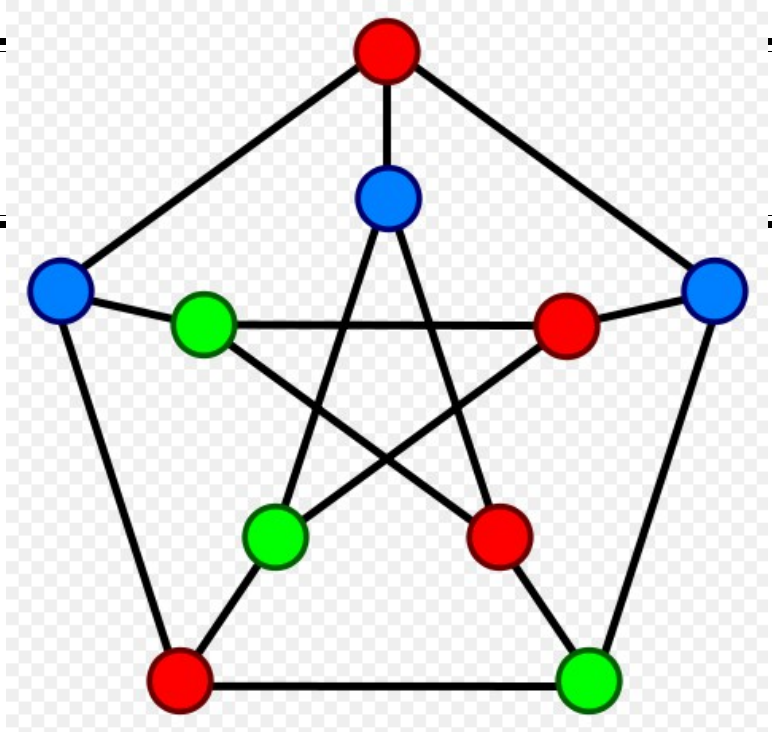
GIÁO TRÌNH

Môn học: Cấu trúc dữ liệu và giải thuật

NGHỀ: QUẢN TRỊ MẠNG

TRÌNH ĐỘ: TRUNG CẤP/CAO ĐẲNG

(Ban hành kèm theo Quyết định số: /2011/QĐ-……của ……



TUYÊN BỐ BẢN QUYỀN:

MÃ TÀI LIỆU: MH17

LỜI GIỚI THIỆU

Kiến thức môn học Cấu trúc dữ liệu và giải thuật là một trong những nền tản cơ bản của những người muốn tìm hiểu sâu về Công nghệ thông tin đặt biệt đối với việc lập trình để giải quyết các bài toán trên máy tính điện tử. Các cấu trúc dữ liệu và các giải thuật được xem như là 2 yếu tố quan trọng nhất trong lập trình, đúng như câu nói nổi tiếng của Niklaus Wirth: Chương trình = Cấu trúc dữ liệu + Giải thuật (Programs = Data Structures + Algorithms). Nắm vững các cấu trúc dữ liệu và các giải thuật là cơ sở để sinh viên tiếp cận với việc thiết kế và xây dựng phần mềm cũng như sử dụng các công cụ lập trình hiện đại.

Cấu trúc dữ liệu có thể được xem như là 1 phương pháp lưu trữ dữ liệu trong máy tính nhằm sử dụng một cách có hiệu quả các dữ liệu này. Và để sử dụng các dữ liệu một cách hiệu quả thì cần phải có các thuật toán áp dụng trên các dữ liệu đó. Do vậy, cấu trúc dữ liệu và giải thuật là 2 yếu tố không thể tách rời và có những liên quan chặt chẽ với nhau. Việc lựa chọn một cấu trúc dữ liệu có thể sẽ ảnh hưởng lớn tới việc lựa chọn áp dụng giải thuật nào.

Về nguyên tắc, các cấu trúc dữ liệu và các giải thuật có thể được biểu diễn và cài đặt bằng bất cứ ngôn ngữ lập trình hiện đại nào. Tuy nhiên, để có được các phân tích sâu sắc hơn và mô phạm, có kết quả thực tế hơn, chúng tôi đã sử dụng ngôn ngữ tựa Pascal để minh hoạ cho các cấu trúc dữ liệu và thuật toán.

Mặc dầu có rất nhiều cố gắng, nhưng không tránh khỏi những khiếm khuyết, rất mong nhận được sự đóng góp ý kiến của độc giả để giáo trình được hoàn thiện hơn.

Đà nẵng, ngày 20 tháng 7 năm 2012

Tham gia biên soạn

1. Ths. Ngô Thị Thanh Trang

2. Ths.Nguyễn Văn Hưng

3. Trương Văn Hòa

MỤC LỤC

ĐỀ MỤC **TRANG**

MÔN HỌC CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Mã môn học: MH17

* VỊ TRÍ, TÍNH CHẤT, Ý NGHĨA VÀ VAI TRÒ CỦA MÔN HỌC

- Vị trí: Môn học được bố trí sau khi sinh viên học xong môn học, mô đun: Lập trình căn bản, Cơ sở dữ liệu.
- Tính chất: Là môn học chuyên ngành bắt buộc
- Ý nghĩa và vai trò: Đây là môn học cơ sở ngành của các ngành liên quan đến công nghệ thông tin, cung cấp cho sinh viên các kiến thức cơ bản về cấu trúc dữ liệu và giải thuật để làm nền tản cho việc lập trình giải quyết các vấn đề cần thiết.

* MỤC TIÊU MÔN HỌC:

- Mô tả được các khái niệm về kiểu dữ liệu trừu tương(danh sách, cây, đồ thị), kiểu dữ liệu, cấu trúc dữ liệu và giải thuật.
- Biết được các phép toán cơ bản tương ứng với các cấu trúc dữ liệu và các giải thuật.
- Biết cách tổ chức dữ liệu hợp lý, khoa học cho một chương trình đơn giản.
- Biết áp dụng thuật toán hợp lý đối với cấu trúc dữ liệu tương ứng để giải quyết bài toán trên máy tính.
- Biết và áp dụng được các phương pháp sắp xếp, tìm kiếm cơ bản
- Bố trí làm việc khoa học đảm bảo an toàn cho người và phương tiện học tập.

*** NỘI DUNG CỦA MÔN HỌC:**

Số TT	Tên các chương trong môn	Thời gian			
		Tổng số	Lý thuyết	Thực hành	Kiểm tra*
1	Tổng quan về Cấu trúc dữ liệu và giải thuật	5	4	1	
2	Đệ qui và giải thuật đệ qui	5	2	2	1
3	Danh sách	30	15	14	1
4	Các phương pháp sắp xếp cơ bản	22	10	11	1
5	Tìm kiếm	8	2	5	1
6	Cây	10	6	4	

7	Đồ thị	10	6	4	
Cộng		90	45	41	4

YÊU CẦU VỀ ĐÁNH GIÁ HOÀN THÀNH MÔN HỌC/MÔ ĐUN

- Về kiến thức: Đánh giá kiến thức qua bài kiểm tra viết, trắc nghiệm đạt được các yêu cầu sau:

  Hiểu được mối quan hệ giữa cấu trúc dữ liệu và giải thuật.

  Phân tích được các kiểu dữ liệu, giải thuật, sự kết hợp chúng để tạo thành một chương trình máy tính.

  Biết cách tổ chức dữ liệu hợp lý, khoa học cho một chương trình đơn giản.

  Biết áp dụng thuật toán hợp lý đối với cấu trúc dữ liệu tương thích để giải quyết bài toán thực tế.

  Biết và áp dụng được các phương pháp sắp xếp, tìm kiếm đơn giản.

- Về kỹ năng:

  Đánh giá kỹ năng thực hành của sinh viên:

  Dùng ngôn ngữ lập trình bất kỳ nào đó thể hiện trên máy tính các bài toán cần kiểm nghiệm về: đệ qui, danh sách, cây, đồ thị, sắp xếp, tìm kiếm...

- Về thái độ: Cẩn thận, tỉ mỉ, thao tác chuẩn xác, tự giác trong học tập.

CHƯƠNG 1: TỔNG QUAN VỀ CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Mã chương: Mh17-01

Giới thiệu:

Tổng quan về giải thuật. Đầu tiên là cách phân tích 1 vấn đề, từ thực tiễn cho tới chương trình, cách thiết kế một giải pháp cho vấn đề theo cách giải quyết bằng máy tính. Tiếp theo, các phương pháp phân tích, đánh giá độ phức tạp và thời gian thực hiện giải thuật cũng được xem xét trong chương.

Mục tiêu:

- Mô tả được khái niệm giải thuật, mối quan hệ giữa cấu trúc dữ liệu và giải thuật. Trình bày được các tiêu chuẩn để đánh giá độ phức tạp của giải thuật.

- Ghi nhớ được các kiểu dữ liệu cơ bản, kiểu dữ liệu trừu tượng và các cấu trúc dữ liệu cơ bản.

- Thực hiện các thao tác an toàn với máy tính.

Nội dung chính:

1.Khái niệm giải thuật và đánh giá độ phức tạp của giải thuật

Mục tiêu: *Mô tả được khái niệm giải thuật, mối quan hệ giữa cấu trúc dữ liệu và giải thuật. Trình bày được các tiêu chuẩn để đánh giá độ phức tạp của giải thuật.*

1.1. Khái niệm giải thuật

Giải thuật, còn gọi là **thuật toán** (algorithm) là một trong những khái niệm quan trọng nhất trong tin học. Thuật ngữ thuật toán xuất phát từ nhà toán học Arập Abu Ja'far Mohammed ibn Musa al Khowarizmi (khoảng năm 825).

Giải thuật thể hiện một giải pháp cụ thể, thực hiện từng bước một để đưa tới lời giải cho một bài toán.

Nói cách khác, **giải thuật** là một tập hữu hạn các phép toán cơ sở, được sắp đặt theo những quy tắc chính xác, nhằm giải một bài toán, hay là một bộ các qui tắc hay qui trình cụ thể nhằm giải quyết một vấn đề trong một số bước hữu hạn, nhằm cung cấp một kết quả từ một tập hợp của các dữ kiện đưa vào.

Các phép toán cơ sở là những phép toán đơn giãn mà thời gian thực hiện nó là hữu hạn và không phụ thuộc vào kích thước của dữ liệu.

Các phép toán trong giải thuật phải được xác định rõ ràng, dễ hiểu, không mập mờ.

Với mọi bộ dữ liệu vào thoả mãn các điều kiện của bài toán, thuật toán phải dừng lại sau một số hữu hạn các bước cần thực hiện

1.2. Ngôn ngữ diễn đạt giải thuật

- Ngôn ngữ tự nhiên.

- Sơ đồ khối.

- Giả ngữ, là một ngôn ngữ "tựa ngôn ngữ lập trình".

- Ngôn ngữ lập trình (Pascal, C,..). Trong tài liệu này chúng ta sử dụng ngôn ngữ tựa Pascal để trình bày. Sau đây là một số qui tắt cơ bản:

1.2.1. Quy cách về cấu trúc chương trình

Mỗi chương trình đều được gán một tên để phân biệt, tên này được viết bằng chữ in hoa, có thể có thêm dấu gạch nối và bắt đầu bằng từ khoá **Program**

Ví dụ : Prorgram NHAN-MA-TRAN

Độ dài tên không hạn chế.

Sau tên có thể kèm theo lời thuyết minh (ở đây ta quy ước dùng Tiếng Việt) để giới thiệu tóm tắt nhiệm vụ của giải thuật hoặc một số chi tiết cần thiết. Phần thuyết minh được đặt giữa hai dấu {...}.

Chương trình bao gồm nhiều bước, mỗi bước được phân biệt bởi số thứ tự, có thể kèm theo những lời thuyết minh.

1.2.2. Kí tự và biểu thức

Kí tự dùng ở đây cũng giống như trong các ngôn ngữ chuẩn, nghĩa là gồm :

26 chữ cái Latinh in hoa hoặc in thường

10 chữ số thập phân

Các dấu phép toán số học: +, - , *, /, (lũy thừa)

Các dấu phép toán quan hệ: <, =, >, , , #.

Giá trị logic: true, false

Dấu phép toán logic: and, or, not

Tên biến là dãy chữ cái và chữ số, bắt đầu bằng chữ cái

Biến chỉ số có dạng : A[i], B[ij] v.v...

Còn biểu thức cũng như thứ tự ưu tiên của các phép toán trong biểu thức cũng theo quy tắc như trong PASCAL hay các ngôn ngữ chuẩn khác.

1.2.3. Các câu lệnh

Các câu lệnh trong chương trình được viết cách nhau bởi dấu chấm phảy chúng bao gồm :

Câu lệnh gán

Có dạng Tên biến/ Tên hàm : = Biểu thức

Ở đây cho phép dùng phép gán chung.

Ví dụ : X : = Y : = 5

Câu lệnh ghép

Có dạng : **begin** Câu lệnh$_1$; Câu lệnh$_2$; ... ; Câu lệnh$_n$ **end**

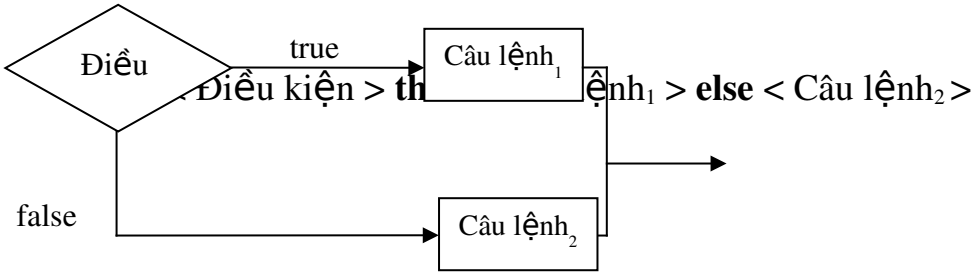
Nó cho phép ghép nhiều câu lệnh lại để được coi như một câu lệnh.

Câu lệnh điều kiện

Có dạng : **if** < Điều kiện > **then < C**âu lệnh >

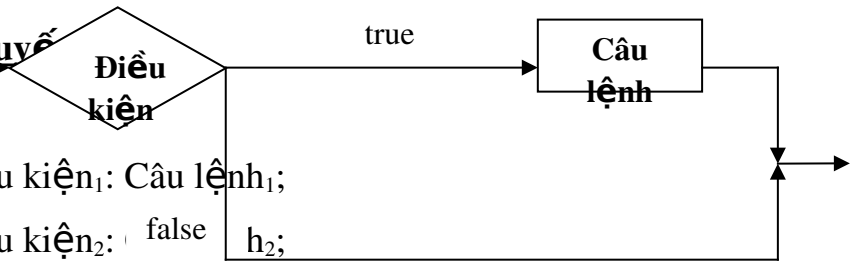
Có thể diễn tả bởi sơ đồ :

Hoặc



< Điều kiện > **th** ệnh$_1$ > **else** < Câu lệnh$_2$ >

Câu lệnh tuyế



Case

Điều kiện$_1$: Câu lệnh$_1$;

Điều kiện$_2$: h$_2$;

.....

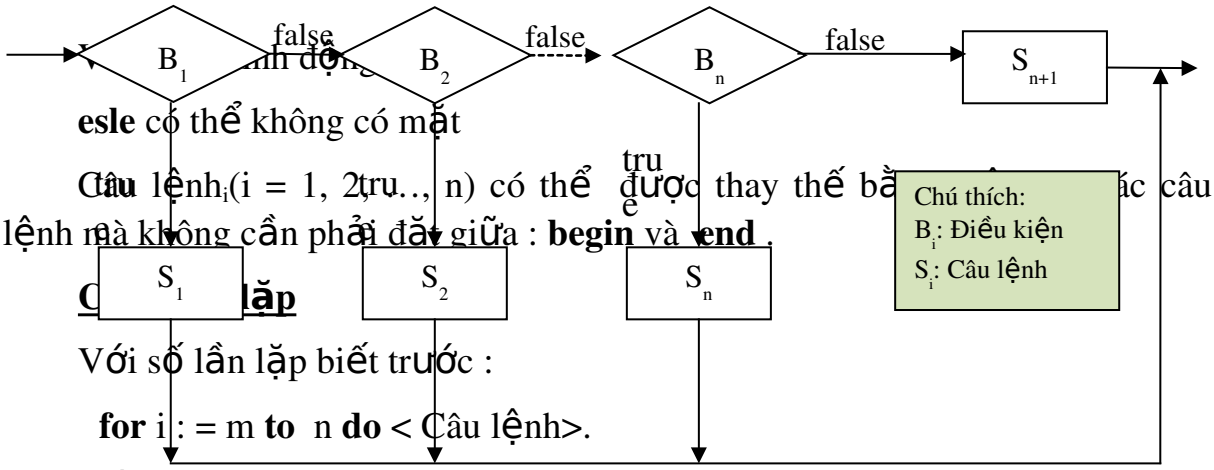
.....

Điều kiện$_n$: Câu lệnh$_n$;

Else: Câu lệnh$_{n+1}$

End case

Câu lệnh này cho phân biệt các tình huống xử lí khác nhau trong các điều kiện khác nhau mà không phải tới các câu lệnh **if – then – else** khác nhau. Có thể diễn tả bởi sơ đồ :



esle có thể không có mặt

Câu lệnh$_i$(i = 1, 2, .., n) có thể được thay thế bằng các câu lệnh mà không cần phải đặt giữa : **begin** và **end** .

Câu lệnh lặp

Với số lần lặp biết trước :

for i : = m **to** n **do** < Câu lệnh>.

nhằm thực hiện < Câu lệnh > với i lấy giá trị nguyên từ m tới n (n m) với bước nhảy tăng bằng 1,

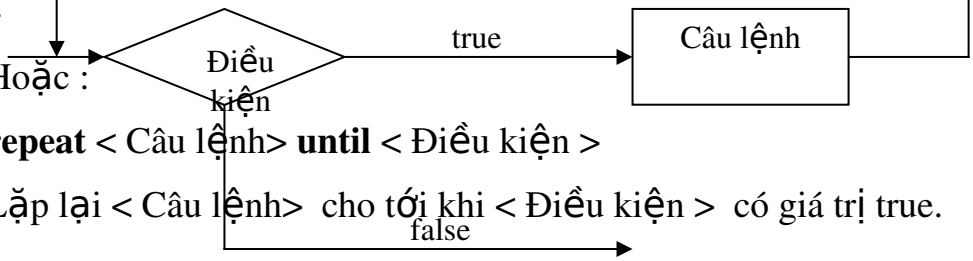
hoặc : **for** i := n **down to** m **do** < Câu lệnh>

tương tự như câu lệnh trên với bước nhảy giảm bằng 1.

Với số lần lặp không biết trước:

While < Điều kiện > **do** < Câu lệnh>

Chừng nào mà < Điều kiện > có giá trị bằng true thì thực hiện < Câu lệnh>



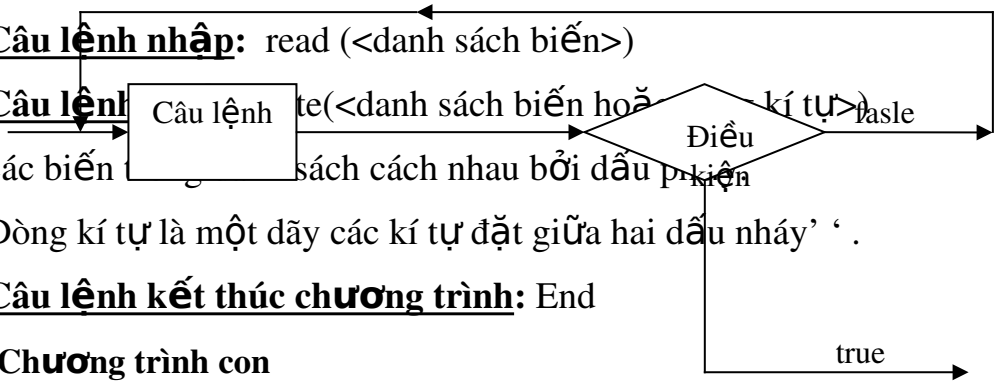
Hoặc :

repeat < Câu lệnh> **until** < Điều kiện >

Lặp lại < Câu lệnh> cho tới khi < Điều kiện > có giá trị true.

Câu lệnh nhập: read (<danh sách biến>)

Câu lệnh te(<danh sách biến hoặ ký tự>)



các biến sách cách nhau bởi dấu

Dòng kí tự là một dãy các kí tự đặt giữa hai dấu nháy’ ‘ .

Câu lệnh kết thúc chương trình: End

1.2.4. Chương trình con

Chương trình con hàm

Có dạng :

Function <tên hàm> (<danh sách tham số>)

S1; S2; … ; Sn.

Return

Chương trình con thủ tục

Có dạng :

Function <tên hàm> (<danh sách tham số>)

S1; S2; … ; Sn.

Return

Câu lệnh kết thúc chương trình ở đây là return thay cho end.

Trong cấu tạo của chương trình con hàm bao giờ cũng có câu lệnh gán mà tên hàm nằm ở vế trái. Còn đối với chương trình con thủ tục thì không có.

Lời gọi chương trình con hàm thể hiện bằng tên hàm cùng danh sách tham số thực sự, nằm trong biểu thức. Còn với chương trình con thủ tục lời gọi được thể hiện bằng câu lệnh call có dạng :

Call <tên thủ tục> (<danh sách tham số thực sự>)

Chú ý : Trong các chương trình diễn đạt một giải thuật ở đây phần khai báo dữ liệu được bỏ qua. Nó được thay với phần mô ta cấu trúc dữ liệu bằng ngôn ngữ tự nhiên, mà ta sẽ nêu ra trước khi bước vào giải thuật.

1.3. Thiết kế giải thuật

Tạo lập giải thuật để giải một bài toán là một nghệ thuật mà không bao giờ có thể nêu đầy đủ ngay một lúc.

Có nhiều phương pháp thiết kế giải thuật khác nhau. Tuy nhiên ta cũng thấy rằng mọi việc sẽ đơn giản hơn nếu như có thể phân chia bài toán lớn thành những bài toán nhỏ hơn, điều đó có nghĩa là có thể coi bài toán của ta như là một Modul chính, cần chia thành các Modul con, và trên tinh thần như vậy đến các modul con ta có thể chia thành các modul nhỏ hơn, chia cho đến khi tới những modul con đủ nhỏ để có thể xử lý trực tiếp. Sau đó chỉ cần tổng hợp lại các phép xử lý để có giải thuật của bài toán gốc.

Để làm được những điều đó, đứng trước một bài toán, thông thường ta phải:

Xác định được rõ dữ liệu và yêu cầu : cho biết cái gì ?(dữ liệu input) và đòi hỏi cái gì ? (dữ liệu output).

Để giải quyết được yêu cầu thì "phải làm gì ?" : ở đây mới chỉ phân hoạch hỏi cái gì ? (dữ liệu output).

Với mỗi công việc ấy thì " phải làm thê nào " ?

Trên cơ sở đó mới cụ thể hóa dần dần các phép xử lí để xây dựng giải thuật cần thiết.

Tất nhiên, khi giải quyết câu hỏi " làm thế nào ?" thì dữ liệu input cũng phải được định hình về cấu trúc.

Ví dụ, ta xét bài toán :

Sắp xếp là một dãy số (a1,a2,….,an) thành một dãy số tăng dần.

Như vậy dãy số input, nếu có dạng, chẳng hạn :

(33, 77, 11, 55, 99, 22, 44, 88, 66)

thì dãy số output phải có dạng :

(11, 22, 33, 44, 55, 66, 77, 88, 99)

Để có được kết quả output như vậy thì phải làm gì ?

Có thể thấy rằng : sắp xếp theo thứ tự tăng dần nghĩa là :

–Số bé nhất trong n số phải được đặt vào vị trí đầu tiên ;

–Số bé nhất trong (n – 1) số còn lại phải được đặt vào vị trí thứ hai ; v.v…

Như vậy sẽ có hai công việc chính phải làm :

Chọn số bé nhất trong dãy số chưa được sắp.

Đặt nó vào vị trí sau phần tử cuối của dãy số đã được sắp (nó lại trở thành phần tử cuối cho bước tiếp theo).

Chú ý rằng : lúc đầu dãy số được sắp còn rỗng, sau đó nó được bổ sung dần dần các phần tử vào.

Các công việc trên sẽ được lặp lại (n - 1) lần : đầu với n số, lần cuối với 2 số.

Để thực hiện được hai công việc nêu trên thì phải "*làm thế nào* ? "

Trước hết phải nghĩ ngay tới : dãy số ở đây được định hình theo cấu trúc nào ? (cấu trúc dữ liệu) và được cài đặt trong máy theo cấu trúc nào ? (mà ta sẽ được gọi là : *cấu trúc lưu trữ*).

Thông thường nó được định hình và cài đặt theo cấu trúc vectơ.

Ở đây có hai vectơ : vectơ input và vectơ output.Vậy thì trong máy ta sẽ dùng hai vectơ để lưu trữ hay chỉ dùng một ?

Giả sử ta chỉ dùng 1, nghĩa là lúc đầu vectơ lưu trữ chứa dãy số cho, nhưng sau khi thực hiện giải thuật thì chính vectơ ấy cũng chứa dãy số đã được sắp xếp(để tiết kiệm bộ nhớ !).

Nếu thế thì công việc "đổi chỗ" sẽ được cụ thể thêm như sau :

–Hoán vị trí của nó (số bé nhất vừa được chọn) với vị trí của số ở đầu dãy chưa được sắp,sau đó gạt nó ra ngoài dãy chưa được sắp(tất nhiên lúc đó nó đã trở thành phần tử cuối của dãy đã được sắp).

Tới đây ta có thể diễn ddajt sơ bộ giải thuật "sắp xếp" của ta như sau :

Procedure SELECTION-SORT(A,n);

{A là vectơ gồm n phần tử là các số cho}

1.{2 công việc được lặp lại (n-1) lần}

for i:=1 **to** (n-1) **do begin**.

2.Chọn số nhỏ nhất A[k] trong dãy các số:

A[i],A[i+1],….,A[n]

3.Hoán vị giữa A[k] và A[i]

4. **return** **end**;

Bây giờ ta đi sâu vào từng công việc :

Làm thế nào để chọn được số nhỏ nhất trong dãy các số:

A[i],A[i+1],….,A[n]?

Có thể tiến hành như sau : thoạt đầu ta cứ chọn A[i],sau đó so sánh các phần tử tiếp theo với nó,nếu phần tử nào nhỏ hơn thì lại thay phần tử đó vào, phần tử cuối cùng được thay chính là phần tử cần tìm.

Nhưng xét cho cùng : ta chỉ cần biết chỉ số k ứng với phần tử nhỏ nhất đó thì sẽ tìm được nó ,vì vậy công việc "chọn" ở trên chỉ cần làm với chỉ số. Có thể diễn đạt như sau :

k:=1 ; { coi phần tử đầu là nhỏ nhất lúc đó,và giữ lại chỉ số của nó}

for j : = i+1 **to** n **do**

if A[j] < A [k] **then** k:=j

Làm thế nào để thực hiện được việc hoán vị chỗ cho hai phần tử ?

Cách giải quyết ở đây giống như khi ta có 2 cốc khác nhau: một đựng rượu, một đựng nước; mà ta lại muốn hoán vị 2 thứ chất lỏng này nghĩa là chuyển sang cốc đang đựng rượu và chuyển rượu sang cốc đang đựng nước.

Rõ ràng điều này chỉ có thể thực hiện được khi ta dùng tới một cóc thứ ba làm cốc trung chuyển.

Từ đó ta có thể diễn đạt việc hoán vị giữa A[k] và A[i] như sau :

LOC : = A[k] ; A[k] := A[i];A[i]:=LOC;

Tổng hợp những ghi nhận ở trên , ta đi tới một thủ tục , thể hiện giải thuật "sắp xếp" của ta ,bằng ngôn ngữ tựa PASCAL như sau :

Procedure SELECTION-SORT (A,n);

1.**for** i:=1 **to** (n-1) **do begin**

2.k:=1;

3.**for** j:=i+1 **to** n **do**

4.**if** A[j] < A[k] **then** k:=j;

5.LOC :=A[k];A[k]:=A[i];A[i]:=LOC

end;

6.**return**

Chú ý:

- ✓ Cách làm ở trên phản ảnh một phương pháp thiết kế giải thuật, gắn liền với lập trình được gọi là "phương pháp tinh chỉnh từng bước" (stepwise refinement).
- ✓ Cách cài đặt một cấu trúc dữ liệu trong máy tính điện tử có thể khác nhau. Vì vậy để phân biệt ta gọi cấu trúc cài đặt trong máy của một "cấu trúc dữ liệu" là "cấu trúc lưu trữ". Như vậy nghĩa là cấu trúc lưu trữ có thể biểu diễn được nhiều cấu trúc dữ liệu khác nhau.

1.4. Đánh giá giải thuật

Khi giải quyết một vấn đề, chúng ta cần chọn trong số các thuật toán, một thuật toán mà chúng ta cho là tốt nhất. Vậy ta cần lựa chọn thuật toán dựa trên cơ sở nào? Thông thường ta dựa trên hai tiêu chuẩn sau đây:

1. Thuật toán đơn giản, dễ hiểu, dễ cài đặt (dễ viết chương trình)

2. Thuật toán sử dụng tiết kiện nhất nguồn tài nguyên của máy tính, và đặc biệt, chạy nhanh nhất có thể được.

Khi ta viết một chương trình chỉ để sử dụng một số ít lần, và cái giá của thời gian viết chương trình vượt xa cái giá của chạy chương trình thì tiêu chuẩn (1) là quan trọng nhất. Nhưng có trường hợp ta cần viết các chương trình (thủ tục hoặc hàm) để sử dụng nhiều lần, cho nhiều người sử dụng, khi đó giá của thời gian chạy chương trình sẽ vượt xa giá viết nó. Chẳng hạn, các thủ tục sắp xếp, tìm kiếm được sử dụng rất nhiều lần bởi rất nhiều người trong các bài toán khác nhau. Trong trường hợp này ta cần dựa trên tiêu chuẩn (2). Ta sẽ cài đặt thuật toán có thể rất phức tạp, miễn là chương trình nhận được chạy nhanh hơn các thuật toán khác.

Tiêu chuẩn (2) được xem là tính hiệu quả của thuật toán. Tính hiệu quả của thuật toán bao gồm hai nhân tố cơ bản

1. Dung lượng không gian nhớ cần thiết để lưu giữ các dữ liệu vào, các kết quả tính toán trung gian và các kết quả của thuật toán.

2. Thời gian cần thiết để thực hiện thuật toán (ta gọi là thời gian chạy chương trình, thời gian này không phụ thuộc vào các yếu tố vật lý của máy tính (tốc độ xử lý của máy tính, ngôn ngữ viết chương trình...))

Chúng ta sẽ chỉ quan tâm đến thời gian thực hiện thuật toán. Vì vậy khi nói đến đánh giá độ phức tạp của thuật toán, có nghĩa là ta nói đến đánh giá thời gian thực hiện. Một thuật toán có hiệu quả được xem là thuật toán có thời gian chạy ít hơn các thuật toán khác.

1.4.1.Đánh giá thời gian thực hiện của giải thuật

Có hai cách tiếp cận để đánh giá thời gian thực hiện của một thuật toán

Phương pháp thử nghiệm: Chúng ta viết chương trình và cho chạy chương trình với các dữ liệu vào khác nhau trên một máy tính nào đó. Thời gian chạy chương trình phụ thuộc vào các nhân tố sau đây:

1. Các dữ liệu vào

2. Chương trình dịch để chuyển chương trình nguồn thành chương trình mã máy.

3. Tốc độ thực hiện các phép toán của máy tính được sử dụng để chạy chương trình.

Vì thời gian chạy chương trình phụ thuộc vào nhiều nhân tố, nên ta không thể biểu diễn chính xác thời gian chạy là bao nhiêuđơn vị thời gian chuẩn, chẳng hạn nó là bao nhiêu giây.

Phương pháp lý thuyết : ta sẽ coi thời gian thực hiện của thuật toán như là hàm số của cỡ dữ liệu vào. Cỡ của dữ liệu vào là một tham số đặc trưng cho dữ liệu vào, no có ảnh hưởng quyết định đến thời gian thực hiện chương trình. Cái mà chúng ta chọn làm cỡ của dữ liệu vào phụ thuộc vào các thuật toán cụ thể. Chẳng hạn, đối với các thuật toán sắp xếp mảng, thì cỡ của dữ liệu vào là số thành phần của mảng; đối với thuật toán giải hệ n phương trình tuyến tính với n ẩn, ta chọn n là cỡ. Thông thường dữ liệu vào là một số nguyên dương n. Ta sẽ sử dụng hàm số T(n), trong đó n là cỡ dữ liệu vào, để biểu diễn thời gian thực hiện của một thuật toán.

Ta có thể xác định thời gian thực hiện T(n) là số phép toán sơ cấp cần phải tiến hành khi thực hiện thuật toán. Các phép toán sơ cấp là các phép toán mà thời gian thực hiện vbị chặn trên bởi một hằng số chỉ phụ thuộc vào cách cài đặt được sử dụng. Chẳng hạn các phép toán số học +, -, *, /, các phép toán so sánh =, <>... là các phép toán sơ cấp.

1.4.2. Độ phức tạp tính toán của giải thuật

Khi đánh giá thời gian thực hiện bằng phương pháp toán học, chúng ta sẽ bỏ qua nhân tố phụ thuộc vào cách cài đặt, chỉ tập trung vào xác định độ lớn của thời gian thực hiện T(n). Ký hiệu toán học O (đọc là ô lớn) được sử dụng để mô tả độ lớn của hàm T(n).

Giả sử n là số nguyên không âm, T(n) và f(n) là các hàm thực không âm. Ta viết T(n) = O(f(n)) (đọc : T(n) là ô lớn của f(n)), nếu và chỉ nếu tồn tại các hằng số dương c và n_0 sao cho T(n) c.f(n), với n > n_0.

Nếu một thuật toán có thời gian thực hiện T(n) = O(f(n)), chúng ta sẽ nói rằng thuật toán có thời gian thực hiện cấp f(n).

Ví dụ : Giả sử $T(n) = 10n^2 + 4n + 4$

Ta có : T(n) $10n^2 + 4n^2 + 4n^2 = 12\ n^2$, với n 1

Vậy $T(n) = O(n^2)$. Trong trường hợp này ta nói thuật toán có độ phức tạp (có thời gian thực hiện) cấp n^2.

Bảng sau đây cho ta các cấp thời gian thực hiện thuật toán được sử dụng rộng rãi nhất và tên gọi thông thường của chúng.

Ký hiệu ô lớn	Tên gọi thông thường
O(1)	Hằng
$O(\log_2 n)$	logarit
O(n)	Tuyến tính
$O(n\log_2 n)$	$n\log_2 n$
$O(n^2)$	Bình phương
$O(n^3)$	Lập phương
$O(2^n)$	Mũ

Danh sách trên sắp xếp theo thứ tự tăng dần của cấp thời gian thực hiện

Các hàm như $\log_2 n$, n, $n\log_2 n$, n^2, n^3 được gọi là các hàm đa thức. Giải thuật với thời gian thực hiện có cấp hàm đa thức thì thường chấp nhận được.

Các hàm như 2n, n!, nn được gọi là hàm loại mũ. Một giải thuật mà thời gian thực hiện của nó là các hàm loại mũ thì tốc độ rất chậm.

2.Các kiểu dữ liệu cơ bản

Mục tiêu: *Ghi nhớ được các kiểu dữ liệu cơ bản.*

Kiểu dữ liệu là một tập hợp các giá trị và một tập hợp các phép toán trên các giá trị đó.

Ví dụ: Kiểu Integer là tập hợp các số nguyên có giá trị từ -32768 đến 32767 cùng các phép toán như {+, -, *, /, div, mod,...}. Kiểu Boolean là một tập hợp gồm 2 giá trị {True, Fasle} và các phép toán trên nó như {and, or, not,...}.

Kiểu dữ liệu sơ cấp là kiểu dữ liệu mà giá trị của nó là đơn nhất.

Thông thường trong một hệ kiểu của ngôn ngữ lập trình sẽ có một số kiểu dữ liệu được gọi là *kiểu dữ liệu sơ cấp* hay *kiểu dữ liệu phân tử* (atomic).

Thông thường, các kiểu dữ liệu cơ bản bao gồm :

Kiểu có thứ tự rời rạc : số nguyên, ký tự, logic, liệt kê, miền con

Kiểu không rời rạc : số thực

Tuỳ từng ngôn ngữ lập trình, các kiểu dữ liệu định nghĩa sẵn này có thể khác nhau đôi chút. Chẳng hạn, với ngôn ngữ C, các kiểu dữ liệu này chỉ gồm số nguyên, số thực, ký tự. Và theo quan điểm của C, kiểu ký tự thực chất cũng là kiểu số nguyên về mặt lưu trữ, chỉ khác về cách sử dụng. Ngoài ra, giá trị logic đúng (TRUE) và giá trị logic sai (FALSE) được biểu diễn trong ngôn ngữ C như là các giá trị nguyên khác 0 và bằng 0. Trong khi đó Pascal định nghĩa tất cả các kiểu dữ liệu đã liệt kê ở trên và phân biệt chúng một cách chặt chẽ.

Sau đây là hệ kiểu của Pascal:

1. Kiểu integer

2. Kiểu real

3. Kiểu boolean

4. Kiểu char

5. Kiểu string

Là kiểu dữ liệu chứa các giá trị là nhóm các ký tự hoặc chỉ một ký tự kể cả chuỗi rỗng. Độ dài tối đa của một biến String là 255 ký tự, tức là nó có thể chứa tối đa một dãy gồm 255 ký tự.

Kiểu dữ liệu String trong pascal được khai báo như sau:

Var $Biến_1$, $Biến_2$,… $Biến_n$: String[số ký tự tối đa]

3.Kiểu bản ghi, kiểu con trỏ

Mục tiêu: *Ghi nhớ được các kiểu dữ liệu bản ghi và kiểu dữ liệu con trỏ*

Kiêu dữ liệu có cấu trúc hay còn gọi là cấu trúc dữ liệu là kiểu dữ liệu mà các dữ liệu của nó là sự kết hợp của các giá trị khác.

Một số kiểu dữ liệu có cấu trúc như: Bản ghi, con trỏ, Array,...

3.1. Kiểu bản ghi

Bản ghi là một cấu trúc bao gồm một số các phần tử có kiểu khác nhau nhưng liên quan với nhau. Các phần tử này gọi là các trường, có thể có những trường trong một bản ghi mà là một bản ghi.

Kiểu dữ liệu bản ghi trong pascal được khai báo như sau:

```
Type
<Tên kiểu> = Record
        <Tên trường 1> : Kiểu;
        <Tên trường 2> : Kiểu;
        ...
        <Tên trường n> : Kiểu;
End;
```

Ví dụ: Khai báo kiểu dữ liệu bảng điểm gồm một số trường nhằm phục vụ quản lý điểm như sau:

```
Type
BangDiem = Record
        Hoten : String[30];
        Lop : String[6];
        Diachi : String;
        DiemLT, DiemTH : real;
End;
```

3.2. Kiểu con trỏ

Khi khai báo một biến mặc nhiên ta qui định độ lớn vùng nhớ dành cho biến.

Ví dụ:

```
Var     x : real;
        y : array[1..50] of integer;
```

như vậy biến a cần 6 byte, biến b cần 100 byte.

Việc khai báo như trên thường là phỏng đoán dung lượng bộ nhớ cần thiết chứ chưa thật sự chính xác. Để tránh lỗi chúng ta thường khai báo dư ra, gây nên lãng phí bộ nhớ. Việc xác định địa chỉ lưu trữ biến và cấp phát bộ nhớ được thực hiện khi biên dịch, nghĩa là các địa chỉ này cũng như dung lượng bộ nhớ cần cấp phát đã được cố định trước khi thực hiện các thao tác khác. Đại lượng này không thay đổi trong suốt quá trình thực hiện chương trình, nói cách khác đây là đại lượng tĩnh.

Để tiết kiệm bộ nhớ, ngay khi chương trình đang làm việc chúng ta có thể yêu cầu cấp phát bộ nhớ cho các biến, điều này được gọi là cấp phát động. Cấp phát bộ nhớ động được thực hiện thông qua biến con trỏ. Muốn có biến con trỏ chúng ta phải định nghĩa kiểu con trỏ trước.

Kiểu con trỏ là một kiểu dữ liệu đặc biệt dùng để biểu diễn các địa chỉ.

Kiểu con trỏ trong Pascal được khai báo như sau:

```
Type
Tên kiểu con trỏ = ^Kiểu dữ liệu;
```

Bài tập thực hành của học viên

1.1.Nêu một vài cấu trúc dữ liệu cơ bản của một ngôn ngữ lập trình mà em biết.

1.2. Khai báo kiểu dữ liệu Nhân sự gồm một số trường: Mã nhân sự, họ tên, lương, địa chi, nhằm phục vụ quản lý nhân sự của một cơ quan.

4.Các kiểu dữ liệu trừu tượng

Mục tiêu: *Ghi nhớ được khái niệm kiểu dữ liệu trừu tượng.*

Kiểu dữ liệu trừu tượng là một mô hình toán học cùng một tập hợp các phép toán trừu tượng được định nghĩa trên mô hình đó. Có thể nói kiểu dữ liệu trừu tượng là một kiểu dữ liệu do chúng ta định nghĩa mức khái niệm, chưa được cài đặt bởi ngôn ngữ lập trình.

Khi cài đặt một kiểu dữ liệu trừu tượng trên một ngôn ngữ lập trình ta thực hiện hai nhiệm vụ:

Biểu diễn kiểu dữ liệu trừu tượng bằng một cấu trúc dữ liệu hoặc bằng một kiểu dữ liệu trừu tượng khác đã được cài đặt.

Viết chương trình con thực hiện các phép toán trên kiểu dữ liệu trừu tượng

Một số kiểu dữ liệu trừu tượng: Danh sách, cây, đồ thị,...

5.Các cấu trúc lưu trữ

Mục tiêu: Ghi *nhớ được các cấu trúc lưu trữ cơ bản: lưu trữ kế tiếp và lưu trữ móc nối* .

5.1. Mảng

5.1.1. Khái niệm

Mảng là một tập hợp có thứ tự, bao gồm một số xác định n phần tử (n được gọi là *độ dài* hay *kích thước* của mảng). Ngoài giá trị, mỗi phần tử của mảng còn được đặc trưng bởi chỉ số, thể hiện thứ tự của phần tử đó trong mảng. Các giá trị của phần tử mảng đều cùng một kiểu dữ liệu.

Vectơ là mảng một chiều, mỗi phần tử của nó ứng với một chỉ số.

Ví dụ: phần tử của vectơ A, kí hiệu là A_i hoặc A[i] với i là chỉ số.

Ma trận là mảng hai chiều, mỗi phần tử của nó ứng với 2 chỉ số.

Ví dụ : phần tử của ma trận B, kí hiệu B_{ij} hoặc B[i,j] với i gọi là chỉ số hàng, j gọi là chỉ số cột.

Tương tự người ta cũng mở rộng : mảng ba chiều, mảng bốn chiều,…. Mảng n chiều.

5.1.2. Cấu trúc lưu trữ của mảng

Một cách đơn giản, có thể hình dung bộ nhớ của máy tính điện tử (MTĐT) là một dãy các phần tử nhớ cơ sở được đánh số kế tiếp nhau (kể từ số 0). Số thứ tự đó được gọi là địa chỉ, môt phần tử nhớ cơ sở, có địa chỉ được gọi là một từ máy. Một phần tử dữ liệu có thể được lưu trữ trong máy bởi một ô nhớ bao gồm một hoặc nhiều từ máy. Việc truy cập vào ô nhớ đó sẽ được xác định bởi địa chỉ của từ máy đầu tiên tạo nên ô nhớ đó. Thường có hai cách để xác định được địa chỉ.

Cách thứ nhất là dựa vào những đặc tả của việc lưu trữ dữ liệu để tính trực tiếp ra địa chỉ. Địa chỉ loại này gọi là địa chỉ được tính. Cách này thường được hay sử dụng trong chương trình dịch của các ngôn ngữ lập trình để tính địa chỉ các phần tử của mảng, tính địa chỉ các lệnh thực hiện tiếp theo v.v …

Cách thứ hai là lưu trữ các địa chỉ cần thiết ở một chổ quy định, khi cần xác định sẽ lấy từ đó ra. Loại địa chỉ này được gọi là con trỏ (pointer) hoặc mối nối (link). Địa chỉ quay lui của chương trình con để quay trở về chỗ có

lời gọi trong chương trình chính, khi kết thúc việc thực hiện chương trình con đó, chính là loại địa chỉ này.

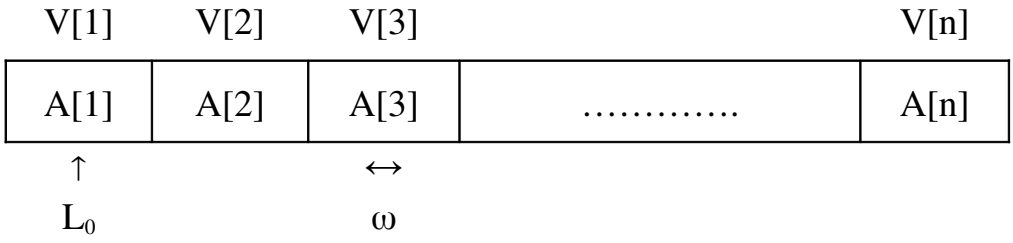
Cũng có một số cấu trúc lưu trữ sử dụng phối hợp cả hai cách xác định địa chỉ nói trên.

Lưu trữ kế tiếp đối với mảng:

Thông thường mảng được lưu trữ trong máy dưới dạng môt vecter lưu trữ. Đó là một dãy các từ máy kế tiếp nhau.

Giả sử, ta xét việc lưu trữ kế tiếp đối với mảng một chiều, hay một vectơ A, mà các phân tử của nó là A[i] với 1 i n. Nếu mỗi phần tử của vectơ được lưu trữ trong một ô nhớ gồm có 1 từ máy thì để lưu trữ vectơ A, phải dành ra trong bộ nhớ n từ máy kế tiếp nhau, đó chính là n phần tử của vectơ đang xét.

Nếu mỗi phần tử của vectơ lưu trữ V (mỗi ô nhớ của V) phải gồm từ máy mới đủ chứa được một phần tử A[i] thì lúc đó V phải bao gồm n* từ máy kế tiếp. Địa chỉ của mỗi ô nhớ, nghĩa là mỗi phần tử nhớ V[i], bây giờ là địa chỉ của từ máy đầu tiên của ô nhớ đó. Ví dụ : nếu =3 mà địa chỉ của V[1] là 1000 thì địa chỉ của V[2] là 1003, của V[3] là 1006.



Hình 2.1

Địa chỉ của V[1] được gọi là địa chỉ gốc (base address), kí hiệu là L_0.

Như vậy việc xác định địa chỉ của V[i], hay nói một cách khác : việc xác định địa chỉ của A[i] sẽ được tính ra theo công thức sau :

$$LOC (A[i]) = L_o + * (i-1)$$

Trong ngôn ngữ như PASCAL, cận dưới của chỉ số không nhất thiết phải là 1, mà có thể là môt số nguyên b nào đó. Khi ấy địa chỉ của A[i] được tính bởi :

$$LOC (A[i]) = L_o + * (i-b)$$

Đối với mảng 2 chiều, hay ma trận, việc lưu trữ các phần tử cũng được thực hiện bởi một vectơ lưu trữ như trên.

Gọi B là một ma trận có m hàng, n cột, B sẽ được lưu trữ trong bộ nhớ bởi vectơ lưu trữ V bao gồm $m*n*$ từ máy (mỗi phần tử của V gồm từ máy).

Nếu giả sử B có 3 hàng, 4 cột (m=3, n=4) thì các phần tử của nó sẽ được lưu trữ như hình sau:

V

B_{11}	B_{12}	B_{13}	B_{14}	B_{21}	B_{22}	B_{23}	B_{24}	B_{31}	B_{32}	B_{33}	B_{34}

V[1] V[5] V[9] V[12]

Hình 2.2

Như vậy nghĩa là : các phần tử của ma trận B sẽ được lưu trữ theo hàng, hết hàng này đến hàng khác.

Cách lưu trữ này được gọi là : *lưu trữ theo thứ tự ưu tiên hàng*

Cũng còn có một cách khác, đó là :lưu trữ theo thứ tự ưu tiên cột. Các phần tử của ma trận sẽ được lưu trữ theo cột, hết cột này đến cột khác.

Với ma trận B[3,4] như trên thì các phần tử sẽ được lưu trữ bởi vectơ lưu trữ V theo hình 2.3 :

V

B_{11}	B_{21}	B_{31}	B_{12}	B_{22}	B_{32}	B_{13}	B_{23}	B_{33}	B_{14}	B_{24}	B_{34}
V[1]			V[4]			V[7]			V[10]		V[12]

Hình 2.3

Việc xây dựng các công thức tính địa chỉ cũng được tiến hành tương tự.

Nếu ma trận B có m hàng, n cột và mỗi phần tử của vectơr lưu trữ V gồm từ máy, thì địa chỉ của B[i,j] với 1 i, j n :

Theo thứ tự ưu tiên hàng sẽ được tính bởi :

$$LOC(B[i,j]) = L_o + [(i - 1) * n + (j - 1)] *$$

Theo thứ tự ưu tiên hàng cột sẽ được tính bởi :

$$LOC(B[i,j]) = L_o + [(j - 1) * m + (i - 1)] *$$

Trường hợp b_1 i u_1, b_2 j u_2 thì mỗi hàng sẽ có $(u_2 - b_2 + 1)$ phần tử. Khi đó công thức tính địa chỉ, chẳng hạn : theo thứ tự ưu tiên hàng, sẽ là :

$$LOC(B[i,j]) = L_o + [(i - b_1) * (u_2 - b_2 + 1) + (j - b_2)] *$$

Người ta cũng mở rộng cách lưu trữ tương tự đối với mảng nhiều chiều.

Chú ý:

Việc truy cập vào một phần tử của mảng được thực hiện một cách trực tiếp thông qua "địa chỉ được tính", nên tốc độ truy cập nhanh và đồng đều đối với mọi phần tử.

Nếu dùng tới cấu trúc mảng trong lập trình thì chúng ta chỉ phải khai báo mảng và làm việc với các tên mảng và biến số. Những vấn đề liên quan đến cấu trúc lưu trữ của mảng cũng như việc xác định địa chỉ để truy cập tới các phần tử mảng mà chúng ta đề cập ở trên đều do chương trình dịch thực hiện.

5.2. Danh sách liên kết

Trong cách tổ chức này, mỗi phần tử của danh sách được lưu trữ trong một ô nhớ mà người ta gọi là "nút"(node). Mỗi nút sẽ bao gồm một số từ máy kế tiếp đủ để lưu trữ các thông tin cần thiết, đó là : thông tin ứng với mỗi phần tử của danh sách và địa chỉ của nút tiếp theo (hay nút kế trước). Với hình thức này các phần tử trong danh sách không cần phải lưu trữ kế cận trong bộ nhớ nên khắc phục được các khuyết điểm của hình thức tổ chức mảng, nhưng việc truy xuất đến một phần tử đòi hỏi phải thực hiện truy xuất qua một số phần tử khác. Có nhiều kiểu tổ chức liên kết giữa các phần tử trong danh sách như :

Danh sách liên kết đơn: mỗi phần tử liên kết với phần tử đứng sau nó trong danh sách:

Như vậy quy cách của mỗi nút có thể hình dung như sau:

INFO	LINK

Nghĩa là mỗi nút gồm có 2 trường.

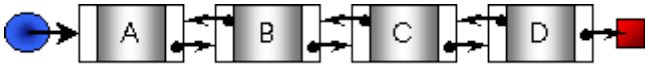
Trường INFO là dữ liệu chứa thông tin ứng với phần tử của danh sách.

Trường LINK có kiểu con trỏ chứa địa chỉ của nút tiếp theo (nút sau nó).

Riêng nút cuối cùng thì không có nút tiếp theo nữa nên trường LINK của nó phải chứa một " địa chỉ đặc biệt", chỉ mang tính chất quy ước, dùng để đánh dấu nút kết thúc danh sách chứ không như các địa chi ở các nút khác, ta gọi nó là "địa chỉ null" hay "mối nối không".

Tất nhiên, để có thể truy cập được vào mọi nút trong danh sách thì phải biết được địa chỉ của nút đầu tiên, hay nói một cách khác là phải "nắm được" con trỏ L, trỏ tới nút đầu tiên này.

Danh sách liên kết kép: mỗi phần tử liên kết với các phần tử đứng trước và sau nó trong danh sách:



Mỗi nút trong danh sách này lại có hai trường con trỏ, theo quy cách như sau:

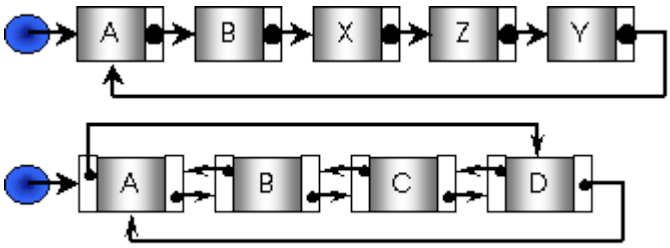
LPTR	INFO	RPTR

Ngoài trường INFO giống như trước đây đã nói có trường LPTR để ghi nhận địa chỉ của nút ở bên trái (nút trước nó) và trường RPTR để ghi nhận địa chỉ nút ở bên phải (nút sau nó).

Như vậy từ một nút không phải chỉ biết địa chỉ của nút sau nó như trong các danh sách nối đơn, mà còn biết cả địa chỉ nút trước nó. Nút đầu tiên không có nút trước nó nên LPTR có giá trị null; đối với nút cuối cùng thì cũng có giá trị null.

Tất nhiên, để có thể truy cập vào danh sách theo cả hai chiều thì phải biết được hai con trỏ: con trỏ L, trỏ tới nút đầu tiên và con trỏ R, trỏ tới nút cuối cùng.

Danh sách liên kết vòng : phần tử cuối danh sách liên kết với phần tử đầu danh sách:



Bài tập thực hành của học viên

1.3.Các cấu trúc dữ liệu cơ bản của một ngôn ngữ lập trình có đủ đáp ứng mọi yêu cầu về tổ chức dữ liệu không?

1.4. Viết công thức tính địa chỉ của phần tử mảng một chiều và mảng hai chiều. Cho mảng AA[15..100], BB[5..30, 7..50], biết địa chỉ gốc L = 500 và mỗi phần tử ứng với $\omega = 4$ từ máy, mảng BB được lưu trữ theo thứ tự ưu tiên hang. Tính AA[55], AA[90], BB[15,15], BB[25,40]

6.Mối quan hệ giữa CTDL và giải thuật

Mục tiêu: *Ghi nhớ được mối quan hệ giữa việc xây dựng cấu trúc dữ liệu và xây dựng giải thuật cho bài toán.*

Thực hiện một đề án tin học là chuyển bài toán thực tế thành bài toán có thể giải quyết trên máy tính. Một bài toán thực tế bất kỳ đều bao gồm các đối tượng dữ liệu và các yêu cầu xử lý trên những đối tượng đó. Vì thế, để xây dựng một mô hình tin học phản ánh được bài toán thực tế cần chú trọng đến hai vấn đề :

Tổ chức biểu diễn các đối tượng thực tế :

Các thành phần dữ liệu thực tế đa dạng, phong phú và thường chứa đựng những quan hệ nào đó với nhau, do đó trong mô hình tin học của bài toán, cần phải tổ chức , xây dựng các cấu trúc thích hợp nhất sao cho vừa có thể phản ánh chính xác các dữ liệu thực tế này, vừa có thể dễ dàng dùng máy tính để xử lý. Công việc này được gọi là xây dựng cấu trúc dữ liệu cho bài toán.

Xây dựng các thao tác xử lý dữ liệu:

Từ những yêu cầu xử lý thực tế, cần tìm ra các giải thuật tương ứng để xác định trình tự các thao tác máy tính phải thi hành để cho ra kết quả mong muốn, đây là bước xây dựng *giải thuật* cho bài toán.

Tuy nhiên khi giải quyết một bài toán trên máy tính, chúng ta thường có khuynh hướng chỉ chú trọng đến việc xây dựng giải thuật mà quên đi tầm quan trọng của việc tổ chức dữ liệu trong bài toán. Giải thuật phản ánh các phép xử lý, còn đối tượng xử lý của giải thuật lại là dữ liệu, chính dữ liệu chứa đựng các thông tin cần thiết để thực hiện giải thuật. Để xác định được giải thuật phù hợp cần phải biết nó tác động đến loại dữ liệu nào và khi chọn lựa cấu trúc dữ liệu cũng cần phải hiểu rõ những thao tác nào sẽ tác động đến nó (ví dụ để biểu diễn các điểm số của sinh viên người ta dùng số thực thay vì chuỗi ký tự vì còn phải thực hiện thao tác tính trung bình từ những điểm số đó). Như vậy trong một đề án tin học, giải thuật và cấu trúc dữ liệu có mối quan hệ chặt chẽ với nhau, được thể hiện qua công thức :

Cấu trúc dữ liệu + Giải thuật = Chương trình

Với một cấu trúc dữ liệu đã chọn, sẽ có những giải thuật tương ứng, phù hợp. Khi cấu trúc dữ liệu thay đổi thường giải thuật cũng phải thay đổi theo để tránh việc xử lý gượng ép, thiếu tự nhiên trên một cấu trúc không phù hợp. Hơn nữa, một cấu trúc dữ liệu tốt sẽ giúp giải thuật xử lý trên đó có thể phát huy tác dụng tốt hơn, giải thuật cũng dễ hiểu và đơn giản hơn.

Ví dụ 1: Một chương trình quản lý điểm thi của sinhviên cần lưu các điểm số của 3 sinh viên. Do mỗi sinh viên có 4 điểm số tương ứng với 4 môn học khác nhau nên dữ liệu có dạng như sau:

Sinh viên	Môn 1	Môn 2	Môn 3	Môn 4
SV1	7	9	7	5
SV2	5	4	2	7
SV3	8	9	6	7

Chỉ xét thao tác xư lý là xuất điểm số các môn của từng sinhviên.

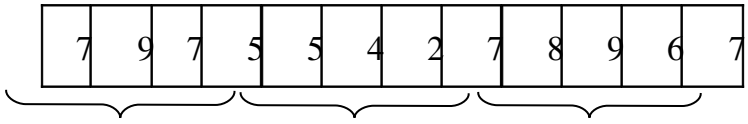
Giả sử có các phương án tổ chức lưu trữ như sau:

Phương án 1: Sử dụng mảng một chiều

có tất cả 3(SV) * 4(Môn) = 12 điểm số cần lưu trữ, do đó ta khai báo mảng như sau:

```
Type mang = array[1..12] of integer;
var  a: mang
```

Khi đó mảng a các phần tử sẽ được lưu trữ như sau:



SV1 SV2 SV3

Và truy xuất điểm số môn j của sinh viên i là phần tử tại dòng i cột j trong bảng. Để truy xuất đến phần tử này ta phải sử dụng công thức xác định chỉ số tương ứng trong mảng a:

Bảng điểm (dòng i, cột j) a[(i -1)*số cột + j]

Ngược lại, với một phần tử bất kỳ trong mảng, muốn biết đó là điểm số của sinh viên nào, môn gì, phải dùng công thức xác định sau:

a[i] bảng điểm (dòng(i div cột) + 1), cột (i mod số cột))

Với phương án này, thao tác xử lý được cài đặt như sau

```
procedure xuat( a: mang);
  var i, mon, so_mon : integer
  begin
        so_mon = 4;
        for  i := 1 to 12 do
```

```
            begin
                sv = i div so_mon;
                mon = i mod so_mon;
                writeln('Điểm môn: ', mon, 'của sinh viên ', sv, ' là: ', a[i]);
            end;
    end;
```

Phương án 2: sử dụng mảng hai chiều

Khai báo mảng hai chiều a có kích thước 3 dòng * 4 cột như sau

```
type mang = array[1..3,1...4] of integer;
var a : mang;
```

	Cột 1	Cột 2	Cột3	Cột 4
Dòng 1	a[1,1] = 7	a[1,2] = 9	a[1,3] = 7	a[1,4] = 5
Dòng 2	a[2,1] = 5	a[2,2] = 4	a[2,3] = 2	a[2,4] = 7
Dòng 3	a[3,1] = 8	a[3,2] = 9	a[3,3] = 6	a[3,4] = 7

Và truy xuất điểm số môn j của sinh viên i là phần tử tại dòng i cột j trong bảng- cũng chính là phần tử ở dòng i cột j trong mảng.

Bảngđiểm (dòng i, cột j) a[i,j]

Với phương án này, thao tác xử lý được cài đặt như sau:

```
procedure xuat( a: mang);
  var i, j, so_sv, so_mon : integer
  begin
        so_mon = 4; so_sv = 3;
        for  i := 1 to  so_sv do
            begin
                for j := 1 to so_mon do
                  writeln('Điểm môn: ', i, 'của sinh viên ', j, ' là: ', a[i,j]);
            end;
  end;
```

Nhận xét

Có thể thấy rõ phương án 2 cung cấp một cấu trúc lưu trữ phù hợp với dữ liệu thực tế hơn phương án 1, và do vậy giải thuật xử lý trên cấu trúc dữ liệu của phương án 2 cũng đơn giản hơn, tự nhiên hơn.

Bài tập thực hành của học viên

1.5.Nêu một giải thuật mà độ phức tạp về thời gian của nó là O(1).

Gợi ý làm bài

1.5. Giải thuật mà độ phức tạp về thời gian của nó là O(1), nếu thời gian thực hiện nó chỉ bằng một hằng số

Ví dụ : giải thuật tính và in X^2

```
Program BINH – PHUONG ;
    Read (X);
    Y = X *X;
    Write Y ;
    Return
```

YÊU CẦU VỀ ĐÁNH GIÁ KẾT QUẢ HỌC TẬP:

Tiêu chí đánh giá	**Kết quả thực hiện**	**Hệ số**	**Kết qủa học tập**
Kiến thức		0,3	
Kỹ năng		0,5	
Thái độ		0,2	
Cộng:			

CHƯƠNG 2: ĐỆ QUY VÀ GIẢI THUẬT ĐỆ QUY

Mã chương: Mh17-02

Giới thiệu:

Đệ qui, một khái niệm rất cơ bản trong toán học và khoa học máy tính. Việc sử dụng đệ qui có thể xây dựng được những chương trình giải quyết được các vấn đề rất phức tạp chỉ bằng một số ít câu lệnh, đặc biệt là các vấn đề mang bản chất truy hồi hạ bậc.

Mục tiêu:

- Trình bày được khái niệm về đệ quy.
- Trình bày được giải thuật và chương trình sử dụng giải thuật đệ quy.
- So sánh giải thuật đệ quy với các giải thuật khác để rút ra tính ưu việt hoặc nhược điểm của giải thuật
- Thực hành (lập trình và biên dịch) với các bài toán đệ quy đơn giản.
- Thực hiện các thao tác an toàn với máy tính.

Nội dung chính:

1.Khái niệm đệ quy

Mục tiêu: *Trình bày được khái niệm về đệ quy.*

Ta nói một đối tượng là đệ quy nếu nó bao gồm chính nó như một bộ phận hoặc nó được định nghĩa dưới dạng của chính nó.

Ví dụ: Trong toán học ta gặp các định nghĩa đệ quy sau:

+ Số tự nhiên:

- 1 là số tự nhiên.
- n là số tự nhiên nếu n-1 là số tự nhiên.

+ Hàm n giai thừa: n!

- 0! = 1
- Nếu n>0 thì N! = n(n-1)!

2.Giải thuật đệ quy và chương trình đệ quy

Mục tiêu: *Trình bày được giải thuật và chương trình sử dụng giải thuật đệ quy.*

2.1. Giải thuật đệ qui

Nếu giải thuật của một bài toán T được thực hiện bằng lời giải của một bài toán T_1 có ý tưởng và nội dung giống bài toán T, nhưng kích thước của tham số bé hơn thì đó là lời giải đệ qui.

2.2. Chương trình đệ qui

Một chương trình con (*hàm hoặc thủ tục*) được gọi là đệ qui nếu trong quá trình thực hiện nó có phần phải gọi tới chính nó.

Trong chương trình con đệ qui có hai phần:

Phần neo(phần dừng): Đặc tả công việc cụ thể cho một hay nhiều tham số.

Phần đệ qui (qui nạp): Công việc ứng với giá trị hiện thời của tham số được định nghĩa bằng công việc ứng với các giá trị khác.

3.Các bài toán đệ quy căn bản

Mục tiêu: *Thực hành (lập trình và biên dịch) với các bài toán đệ quy đơn giản.*

3.1. Bài toán tính n giai thừa

Hàm này được định nghĩa như sau:

$$Factorial(n) = \begin{cases} 1 & \text{nếu } n = 0 \\ n * Factorial(n-1) & \text{nếu } n > 0 \end{cases}$$

Giải thuật đệ quy được viết dưới dạng hàm dưới đây:

```
Function  Factorial(n)
  Begin
        if n=0 then Factorial:=1
        else    Factorial := n*Factorial(n-1);
  End;
```

Trong hàm trên lời gọi đến nó nằm ở câu lệnh gán sau else.

Mỗi lần gọi đệ quy đến Factorial, thì giá trị của n giảm đi 1. Ví du, Factorial(4) gọi đến Factorial(3), gọi đến Factorial(2), gọi đến Factorial(1), gọi đến Factorial(0) đây là trường hợp suy biến, nó được tính theo cách đặc biệt Factorial(0) = 1.

3.2. Bài toán dãy số FIBONACCI.

Dãy số Fibonacci bắt nguồn từ bài toán cổ về việc sinh sản của các cặp thỏ. Bài toán được đặt ra như sau:

- Các con thỏ không bao giờ chết.

- Hai tháng sau khi ra đời một cặp thỏ mới sẽ sinh ra một cặp thỏ con.

- Khi đã sinh con rồi thì cứ mỗi tháng tiếp theo chúng lại sinh được một cặp con mới.

Giả sử bắt đầu từ một cặp thỏ mới ra đời thì đến tháng thứ n sẽ có bao nhiêu cặp?

Ví dụ với n = 6, ta thấy.

Tháng thứ 1: 1 cặp (cặp ban đầu)

Tháng thứ 2: 1 cặp (cặp ban đầu vẫn chưa đẻ)

Tháng thứ 3: 2 cặp (đã có thêm 1 cặp con)

Tháng thứ 4: 3 cặp (cặp đầu vẫn đẻ thêm)

Tháng thứ 5: 5 cặp (cặp con bắt đầu đẻ)

Tháng thứ 6: 8 cặp (cặp con vẫn đẻ tiếp)

Đặt F(n) là số cặp thỏ ở tháng thứ n. Ta thấy chỉ những cặp thỏ đã có ở tháng thứ n-2 mới sinh con ở tháng thứ n do đó số cặp thỏ ở tháng thứ n là:

F(n) = f(n-2) + F(n-1) vì vậy F(n) có thể được tính như sau:

$$F(n) = \begin{cases} 1 & \text{nếu } n \le 2 \\ F(n-2) + F(n-1) & \text{nếu } n > 2 \end{cases}$$

Dãy số thể hiện F(n) ứng với các giá trị của n = 1, 2, 3, 4..., có dạng

1 1 2 3 5 8 13 21 34 55.... nó được gọi là dãy số Fibonacci. Nó là mô hình của rất nhiều hiện tượng tự nhiên và cũng được sử dụng nhiều trong tin học.

Sau đây là thủ tục đệ quy thể hiện giải thuật tính F(n).

```
Function F(n)
  Begin
        if n<=2 then F:=1
        else F := F(n-2) + F(n-1);
  End;
```

ở đây trường hợp suy biến ứng với 2 giá trị F(1) = 1 và F(2) = 1.

Bài tập thực hành của học viên

2.1. Giả sử a và b là những số nguyên dương. Q là hàm số của a,b, được định nghĩa như sau:

Q(a,b)= $\begin{cases} 0, & a = b \\ Q(a-b,b)+1, & a > b \end{cases}$

Hãy tính Q(2,3) và Q(14,3)

2.2. Cho biết số Fibonacci $F_{11} = 89$ và $F_{12} = 144$

a. Hãy tính F_{16}.

b. Viết một thủ tục không đệ quy (dùng phép lặp) để tính và in ra n số Fibonacci đầu tiên.

2.3. Giải thuật tính ước số chung lớn nhất của 2 số p và q (p > q) được mô ta như sau (giải thuật Euclide)

Gọi r là số dư trong phép chia p cho q :

- Nếu r = 0 thi q là ước số chung lớn nhất
- Nếu r 0 thì gán cho p giá trị của q , gán cho q giá trị của r rồi lặp lại quá trình trên.

a. Hãy lập bảng ghi nhận các giá trị của p, q, r trong quá trình thực hiện tính ước số chung lớn nhất của 2 số : 1260 và 198.

b. Hãy nêu lên tính đệ quy trong cách tính này từ đó xây dựng một cách tính đệ quy cho hàm tính ước số chung lớn nhất

USCLN (p,q)

c. Viết một giải thuật đệ quy và một giải thuật không đệ quy (dùng phép lặp) để tính ước số chung lớn nhất của p,q

Gợi ý làm bài

2.2. a. $F_{16} = 987$

b. Procedure FIBONACCI (n, F);

{ở đây F là một vecto có n phần tử để lưu trữ n số Fibonacci đầu tiên}

B1: {Tạo lập 2 số Fibonacci đầu tiên}

F[1] := 1 ; F[2]:= 1;

B2: {Tính lần lượt các số Fibonacci tiếp theo}

For i:=3 to n do

F[i] := F[i-1] + F[I – 2];

B3: {in lần lượt n số Fibonacci}

For i:=1 to n do

Write (F[i]);

B4: Return

2.3. a. Các giá trị của p,q,r được ghi nhận trong bảng sau:

p	q	r
1260	198	72
198	72	54
72	54	18
54	18	0

cuối cùng ta có : USCLN (1260, 198) = 18

b. Ta thấy việc tính USCLN của p và q sẽ dẫn tới việc tính USCLN của q và r khi r # 0 (q và r rõ ràng là nhỏ hơn p và q).

c. Giải thuật đệ quy:

```
Function RUSCLN (p,q);
{Chú ý là số dư r của p và q được xác định bởi phép tính p mod q}
B1: if p mod q = 0 then RUSCLN := q
B2:                else RUSCLN := RUSCLN (q,p mod q);
B3: return
```

Giải thuật lặp

```
Function IUSCLN (p,q);
{x, y, z ở đây là các biến cục bộ}
B1: x := p;    y:=q;
B2 : repeat     z := x mod y;
                    x := y;
                    y := z;
       until  z := 0;
B3: IUSCLN := x;
B4: return.
```

Yêu cầu về đánh giá kết quả học tập:

- Đưa ra các nội dung, sản phẩm chính...;

- Cách thức và phương pháp đánh giá...;

- Gợi ý tài liệu học tập..;

Ghi nhớ

- .....

- .....

CHƯƠNG 3: DANH SÁCH

Mã chương: Mh17-03

Giới thiệu:

Danh sách là cấu trúc dữ liệu rất thông dụng được cài đặt trên mảng và danh sách liên kết, ngăn xếp và hàng đợi. Đó là các cấu trúc dữ liệu cũng rất gần gũi với các cấu trúc trong thực tiễn.

Mục tiêu:

- Trình bày khái niệm và các phép toán cơ bản trên danh sách;
- Biết các cấu trúc cài đặt cho danh sách và các phép toán tương ứng với các cấu trúc dữ liệu;
- Giải được các bài toán sử dụng danh sách.
- Thực hiện các thao tác an toàn với máy tính.

Nội dung chính:

1.Danh sách và các phép toán cơ bản trên danh sách

1.1. Khái niệm danh dách

Có thể nói : Trong công việc hàng ngày, danh sách là loại rất phổ dụng : danh sách các lớp nghề quản trị mạng , danh sách các sinh viên tham gia văn nghệ v.v..

Tất cả danh sách đều có một điểm chung là : Danh sách bao gồm một số hữu hạn phần tử, có thứ tự và số lượng phần tử có thể biến động.

Có thể hình dung : danh sách A là một dãy các phần tử : $(a_1,a_2,...,a_n)$ với n là một biến.

Vectơ chính là hình ảnh của một danh sách tại một thời điểm nào đó.

Trong một danh sách luôn có các phần tử đầu (phần tử thứ nhất), phần tử cuối (phần tử thứ n). Với mỗi phần tử, có phần tử trước nó (trừ phần tử đầu) và phần tử sau nó (trừ phần tử cuối).

1.2. Các phép toán trên danh dách

Đối với danh sách thì thường có phép khởi tạo danh sách rỗng, kiểm tra danh sách rỗng, chèn phần tử vào danh sách, xóa phần tử khỏi danh sách. Ngoài ra có thể còn có các phép như:

Tìm kiếm một phần tử theo một tiêu chí xác định

Sắp xếp các phần tử theo một thứ tự ấn định

Ghép hai hoặc nhiều danh sách thành một danh sách lớn.

Tách một danh sách thành nhiều danh sách con v.v…

2.Cài đặt danh sách theo cấu trúc mảng

Chúng ta có thể cài đặt danh sách bằng mảng như sau: *dùng một mảng để lưu giữ liên tiếp các* phần *tử của danh sách từ vị trí đầu tiên của mảng*. Với cách cài đặt này, ta phải ước lượng số phần tử của danh sách để khai báo số phần tử của mảng cho thích hợp. Dễ thấy rằng số phần tử của mảng phải được khai báo không ít hơn số phần tử của danh sách. Nói chung là mảng còn thừa một số chỗ trống. Mặt khác ta phải lưu giữ độ dài hiện tại của danh sách, độ dài này cho biết danh sách có bao nhiêu phần tử và cho biết phần nào của mảng còn trống.

Chúng ta mô tả danh sách được cài đặt bằng mảng như sau:

Chỉ số	1	2	…	Last	...	Maxlength
Nội dung phần tử	Phần tử thứ 1	Phần tử thứ 2	…	Phần tử cuối cùng trong danh sách	...	

Ta xem vị trí của một phần tử trong danh sách là chỉ số của mảng tại vị trí lưu trữ phần tử đó.

<u>Các khai báo cần thiết là</u>:

```
Const   MaxLength ...
{Số nguyên thích hợp để chỉ độ dài của danh sách}
Type
 ...  Kieuphantu; {kiểu của phần tử trong danh sách}
Type
  Position = integer; { Kiểu vị trí của các phần tử }
Type
Tenkieumang = ARRAY [Chỉ số] OF Kieuphantu;
{mảng chứa các phần tử của danh sách }
Position Last; {giữ độ dài danh sách }
end;
```

Trên đây là sự biểu diễn kiểu dữ liệu trừu tượng danh sách bằng cấu trúc dữ liệu mảng. Phần tiếp theo là cài đặt các phép toán cơ bản trên danh sách.

2.1. Khởi tạo danh sách rỗng

Danh sách rỗng là một danh sách không chứa bất kỳ một phần tử nào (hay độ dài danh sách bằng 0). Theo cách khai báo trên, trường Last chỉ vị trí của phần tử cuối cùng trong danh sách và đó cũng độ dài hiện tại của danh sách, vì vậy để khởi tạo danh sách rỗng ta chỉ việc gán giá trị trường Last này bằng 0.

```
Procedure MakeNull_List;
 Begin
      Last:=0;
 End;
```

2.2. Kiểm tra danh sách rỗng

Danh sách rỗng là một danh sách mà độ dài của nó bằng 0.

```
Function Empty_List;
 Begin
      return  Last=0;
 End;
```

2.3. Chèn phần tử vào danh sách

Khi chèn phần tử có nội dung x vào tại vị trí p của danh sách L thì sẽ xuất hiện các khả năng sau:

- Mảng đầy: mọi phần tử của mảng đều chứa phần tử của danh sách, tức là phần tử cuối cùng của danh sách nằm ở vị trí cuối cùng trong mảng. Nói cách khác, độ dài của danh sách bằng chỉ số tối đa của mảng; Khi đó không còn chỗ cho phần tử mới, vì vậy việc xen là không thể thực hiện được, chương trình con gặp lỗi.

- Ngược lại ta tiếp tục xét:

Nếu p không hợp lệ (p>last+1 hoặc p<1) thì chương trình con gặp lỗi; (Vị trí xen p<1 thì khi đó p không phải là một vị trí phần tử trong trong danh sách đặc. Nếu vị trí p>L.last+1 thì khi xen sẽ làm cho danh sách L không còn là một danh sách đặc nữa vì nó có một vị trí trong mảng mà chưa có nội dung.)

Nếu vị trí p hợp lệ thì chúng ta tiến hành xen theo các bước sau:

+ Dời các phần tử từ vị trí p đến cuối danh sách ra sau 1 vị trí.

+ Độ dài danh sách tăng 1.

+ Đưa phần tử mới vào vị trí p

Chương trình con chèn phần tử x vào vị trí p của danh sách L có thể viết như sau:

```
Procedure Insert_List(X : Kieuphantu, P : Position);
  Var Q : Position;
  Begin
        if (Last=MaxLength) then
              Write("Danh sach day");
        else if ((P < 1) or (P > Last))
              Write("Vi tri khong hop le");
        Else
        begin
        {Dời các phần tử từ vị trí p đến cuối danh sách sang phải 1 vị trí}
              For  Q:=Last  downto P do
                    Elements[Q]:=Elements[Q-1];
              {Đưa X vào vị trí P}
              Elements[P]=X;
              {Tăng độ dài danh sách lên 1 }
              Last:=Last + 1;
        End;
  End;
```

2.4. Xóa phần tử khỏi danh sách

Xoá một phần tử ở vị trí p ra khỏi danh sách L chúng ta làm công việc ngược lại với xen.

- Trước tiên, kiểm tra vị trí phần tử cần xóa xem có hợp lệ hay chưa. Nếu p>L.last hoặc p<1 thì đây không phải là vị trí của phần tử trong danh sách.

- Ngược lại, vị trí đã hợp lệ thì phải dời các phần tử từ vị trí p+1 đến cuối danh sách ra trước một vị trí và độ dài danh sách giảm đi 1 phần tử (do đã xóa bớt 1 phần tử).

```
Procedure Delete_List(P : Position);
  Var Q : Position;
  Begin
        if ((P<1) or (P>Last)) then
              Write ("Vi tri khong hop le");
        else if (Last<1)
              Write ("Danh sach rong!");
        Else
        Begin
```

{Dời các phần tử từ vị trí p+1 (chỉ số trong mảng là p) đến cuối danh sách sang trái 1 vị trí}

```
              For Q:=P to Last do
                    Elements[Q]:=Elements[Q+1];
              Last:=Last - 1;
        End;
  End;
```

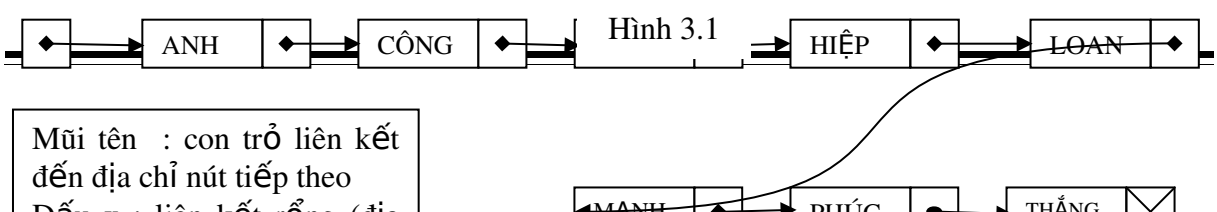
3.Cài đặt danh sách theo cấu trúc danh sách liên kết (đơn, kép)

Như đã nêu ở trên, lưu trữ kế tiếp đối với danh sách tuyến tính đã thể hiện rõ nhược điểm trong trường hợp thực hiện thường xuyên các phép bổ sung hoặc loại bỏ phần tử, trường hợp xử lý đồng thời nhiều danh sách v.v...

Việc sử dụng cấu trúc dữ liệu danh sách liên kết để cài đặt kiểu dữ liệu trừu tượng danh sách chính là một giải pháp nhằm khắc phục nhược điểm trên.

Để có thể truy nhập vào mọi nút trong danh sách, ta phải truy nhập từ nút đầu tiên, nghĩa là cần có một con trỏ head trỏ tới nút đầu tiên này.

Có thể minh họa danh sách móc nối này bằng hình ảnh như sau :



Hình 3.1

chỉ null)

Dưới đây là khai báo cấu trúc dữ liệu biểu diễn danh sách liên kết đơn.

```
Type
  Pointer = ^Node ;
                    Node = Record
                     Info : Kieuphantu ;
                     Link: Pointer;
  End ;
```

Ví dụ : Cấu trúc node "SinhVien" như sau:

```
Type    pSinhVien = ^SinhVien
        SinhVien=Record
                MaSV :      String[10];
                Ten:   String[7];
                Diem1,Diem2: Integer;
                DTB: Real;
                Link: PsinhVien;
End;
```

Câu lệnh **call** New(p) sẽ cho ta một nút trống với quy cách ấn định, có địa chỉ là p để sử dụng; còn câu lệnh **call** dispose(p) sẽ trả lại cho "danh sách chỗ trống" nút địa chỉ là p

Sau đây ta sẽ xét tới một số giải thuật thực hiện một số phép xử lý trên danh sách móc nối.

3.1. Khởi tạo danh sách rỗng

Để khởi tạo danh sách rỗng ta chỉ cần lệnh gán:

```
head := NIL;
```

3.2. Kiểm tra danh sách rỗng

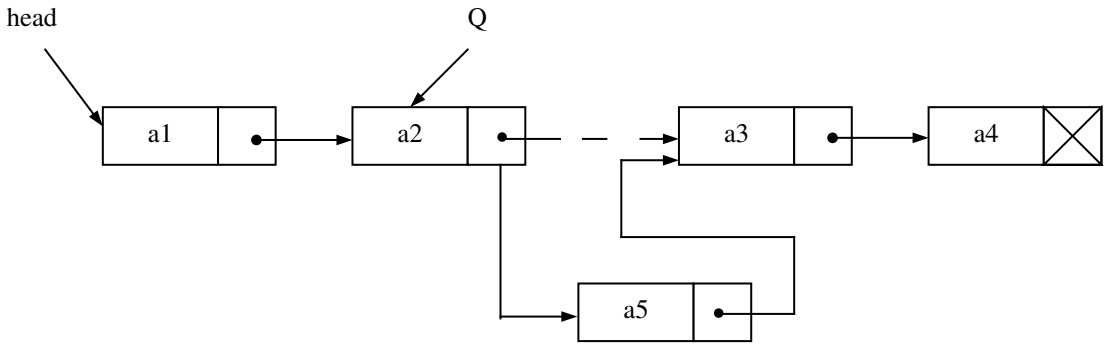
Điều kiện để danh sách liên kết đơn rỗng là head = NIL.

3.3. Chèn phần tử vào danh sách

Giả sử Q là một con trỏ trỏ vào một nút của danh sách, ta cần bổ sung một nút mới với info là X vào sau nút được trỏ bởi Q. Phép toán này được thực hiện bởi thủ tục sau:

```
Procedure InsertAfter(Var head : Pointer; Q : Pointer; X : Kieuphantu);
  Var  P : Pointer;
  Begin
  {1. Tạo một nút mới}
        NEW(P);
        P^.info := X;
  {Thực hiện bổ sung, nếu danh sách rỗng thì bổ sung nút mới vào thành
nút đầu tiên, ngược lại bổ sung nút mới vào sau nút được trỏ bởi Q}
        If head = NIL Then
              begin
                    P^.Link := NIL;
                    head := P;
              end
        Else
              begin
                    P^.Link := Q^.Link;
                    Q^.Link := P;
              end;
  End;
```

Có thể mô tả thao tác bổ sung trên bằng hình sau.



Hình 3.2

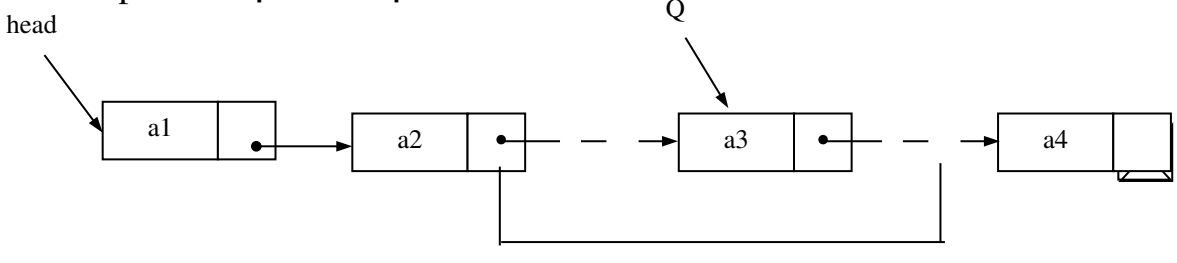
3.4. Xóa phần tử khỏi danh sách

Cho danh sách liên kết đơn được trỏ bởi head. Q là một con trỏ trỏ vào một nút trong danh sách. Giả sử ta cần loại bỏ nút được trỏ bởi Q. ở đây ta cũng gặp khó khăn là nếu Q không phải là nút đầu tiên thì không xác định được nút đứng trước Q. Trong trường hợp này ta phải tìm đến nút đứng trước

Q và cho con trỏ R trỏ vào nút đó, tức là Q = R^.Link. Sau đó ta mới thực hiện loại bỏ nút Q. Ta có thủ tục sau:

```
Procedure Delete(Var head:Pointer; Q:Pointer; Var X:Kieuphantu);
Var R : Pointer;
Begin
{1. Trường hợp danh sách rỗng}
  If head = NIL Then
        begin
              writeln('Danh sach rong');
              Exit;
        end;
  X := Q^.info; {lưu thông tin của nút cần loại bỏ vào biến X}
{2. Trường hợp nút trỏ bởi Q là nút đầu tiên}
  If Q = head Then
        begin
              head := Q^.Link;
              DISPOSE(Q);
              Exit;
        end;
  {3. Tìm đến nút đứng trước nút trỏ bởi Q}
  R := head;
  While R^.Link <> Q Do R := R^.Link;
  {4. Loại bỏ nút trỏ bởi Q}
  R^.Link := Q^.Link;
  DISPOSE(Q);
End;
```

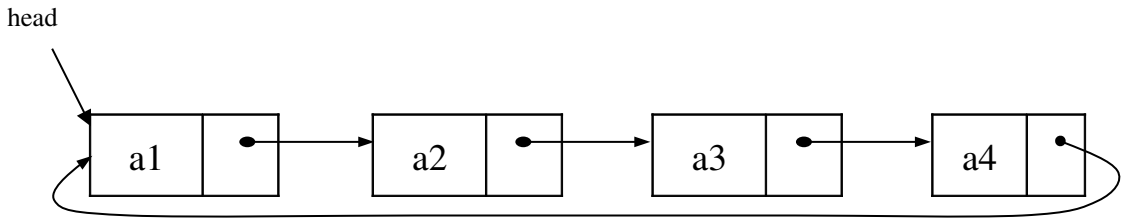
Phép toán loại bỏ được mô tả bởi hình sau:



Hình 3.3

3.5. Danh sách liên kết vòng

Một cải tiến của danh sách liên kết đơn là kiểu danh sách liên kết vòng. Nó khác với danh sách liên kết đơn ở chỗ trường Link của nút cuối cùng trong danh sách không phải bằng NIL, mà nó trỏ đến nút đầu tiên trong danh sách, tạo thành một vòng tròn. Hình ảnh của nó như sau:

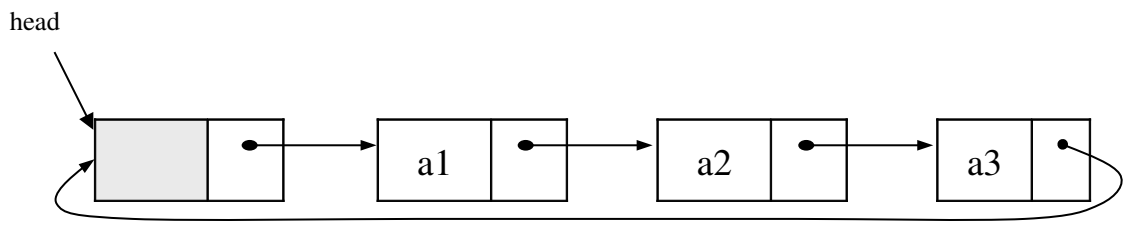


Hình 3.4

Cải tiến này làm cho việc truy nhập vào các nút trong danh sách được linh hoạt hơn. Ta có thể truy nhập vào mọi nút trong danh sách bắt đầu từ nút nào cũng được, không nhất thiết phải từ nút đầu tiên. Điều đó có nghĩa là nút nào cũng có thể coi là nút đầu tiên và con trỏ Head trỏ tới nút nào cũng được. Như vậy, đối với danh sách liên kết vòng chỉ cần cho biết con trỏ trỏ tới nút muốn loại bỏ ta vẫn thực hiện được vì vẫn tìm được đến nút đứng trước đó. Với phép ghép, phép tách cũng có những thuận lợi nhất định.

Tuy nhiên, danh sách nối vòng có một nhược điểm rất rõ là trong khi xử lý, nếu không cẩn thận sẽ dẫn tới một chu trình không kết thúc, bởi vì không biết được vị trí kết thúc danh sách.

Để khắc phục nhược điểm này, người ta đưa thêm vào danh sách một nút đặc biệt gọi là "nút đầu danh sách". Trường info của nút này không chứa dữ liệu của phần tử nào và con trỏ Head bây giờ trỏ tới nút đầu danh sách này. Việc dùng thêm nút đầu danh sách đã khiến cho danh sách về mặt hình thức không bao giờ rỗng. Hình ảnh của nó như sau:



Hình 3.5

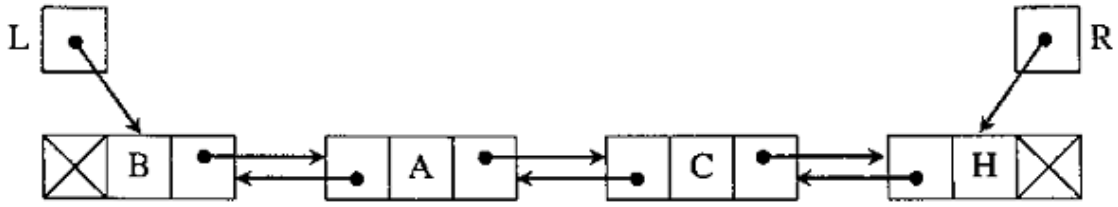
Sau đây là đoạn giải thuật bổ sung một nút vào thành nút đầu tiên trong danh sách có "nút đầu danh sách" trỏ bởi head.

```
New(P);
P^.infor := X;
P^.Link := Head^.Link;
Head^.Link := P;
```

3.6. Danh sách liên kết đôi

LLink	Info	RLink

Khi làm việc với danh sách, có những xử lý trên mỗi nút của danh sách lại liên quan đến cả nút đứng trước và nút đứng sau. Trong những trường hợp như thế, để thuận tiện, người ta đưa vào mỗi nút của danh sách hai con trỏ: LLink trỏ đến nút dứng trước và RLink trỏ đến nút đứng sau nó. Để truy nhập vào danh sách ta dùng hai con trỏ: con trỏ L trỏ vào nút đầu tiên và con trỏ R trỏ vào nút cuối cùng của danh sách. Hình ảnh của danh sách liên kết đôi như sau:



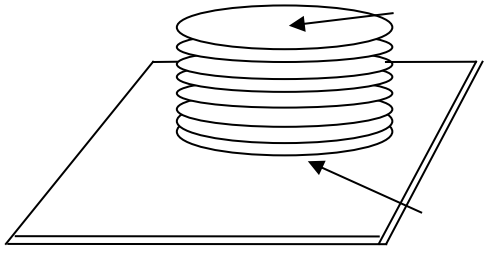
Hình 3.6

4. Danh sách đặc biệt

4.1. Ngăn xếp

Ngăn xếp (Stack) là một kiểu danh sách đặc biệt mà phép bổ sung và phép loại bỏ luôn thực hiện ở một đầu ; được gọi là đỉnh.

Có thể hình dung cách tổ chức lưu trữ của stack như một chồng đĩa đặt trên bàn. Đặt thêm một đĩa mới vào thì đặt phía trên đỉnh, lấy một đĩa ra khỏi chồng thì cũng phải lấy ra từ đỉnh. Đĩa đưa vào sau cùng, chính là đĩa đang nằm ở đỉnh, và nó cũng chính là đĩa sẽ lấy ra trước tiên lại đang ở vị trí được gọi là đáy và nó chính là đĩa được lấy ra sau cùng.



Hình 3.7

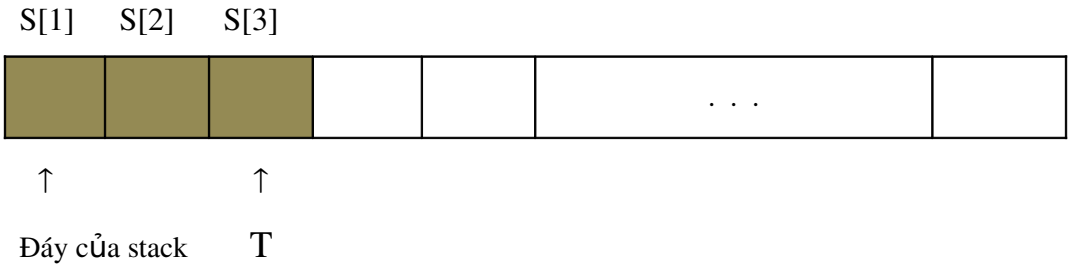
Như vậy stack còn được gọi là danh sách kiểu LIFO (last – in –first –out), tức là stack hoạt động theo cơ chế : "vào – sau – ra – trước"

Biểu diễn stack bằng mảng:

Dùng một mảng để lưu trữ liên tiếp các phần tử của stack. Các phần tử được đưa vào stack bắt đầu từ chỉ số cao nhất của mảng. Chúng ta dùng một biến số nguyên T để lưu trữ chỉ số của phần tử tại đỉnh stack.

Chúng ta quy ước T = 0 nghĩa là stack rỗng. Như vậy T = i thì stack có i phần tử. Rõ ràng 0 ≤ T ≤ n, khi T = n thì stack đã đầy, lúc đó nếu có phép bổ sung một phần tử mới vào stack thì sẽ không thực hiện được, vì "không còn chỗ" ; ta nói là có hiện tượng "tràn" và tất nhiên việc xử lí phải ngừng lại. Còn nếu T =0, nghĩa là stack đã rỗng, mà lại có phép loại bỏ một phần tử ra khỏi stack thì phép xử lí này cũng không thực hiện được; Ta nói có hiện tượng "cạn"

Sau đây là hình ảnh cài đặt của stack với 3 phần tử.



Khi bổ sung một phần tử mới vào thì T tăng lên 1, còn khi loại bỏ một phần tử ra khỏi stack thì T giảm đi 1.

Chúng ta khai báo cấu trúc dữ liệu biểu diễn ngăn xếp như sau:

Const max = N ;

Type Stack =Array[1..max] Of Kieuphantu;

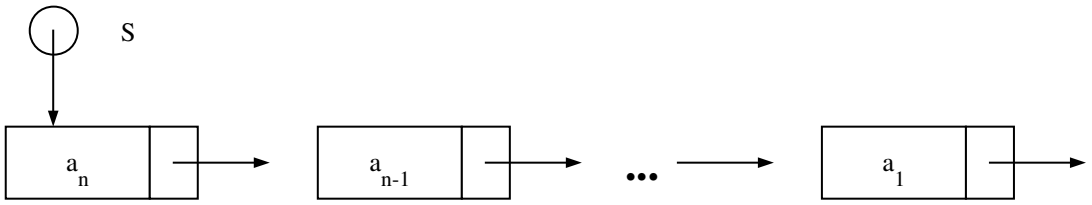
End ;

Var S : Stack ;

Biểu diễn stack bằng danh sách liên kết:

Đối với stack việc truy cập chỉ được thực hiện 1 đầu (đỉnh). Vì vậy, việc cài đặc stack bằng một danh sách nối đơn, có con trỏ head trỏ tới nút đầu tiên, là một cách biểu diễn rất phù hợp. Chúng ta có thể coi head như con trỏ đang trỏ tới đỉnh stack. Bổ sung một nút vào stack chính là bổ sung một nút vào để nó trở thành nút đầu tiên của danh sách, loại bỏ một nút ra khỏi stack chính là loại bỏ nút đầu tiên của danh sách, đang trỏ bởi head. Trong việc bổ sung với ngăn xếp dạng này không cần kiểm tra hiện tượng tràn như với ngăn xếp lưu trữ kế tiếp.

Để cài đặt ngăn xếp bởi danh sách liên kết, ta sử dụng con trỏ S trỏ vào đỉnh của ngăn xếp (hình 3.8).



Hình 3.8

Cấu trúc dữ liệu của ngăn xếp được khai báo như sau:

```
Type    pointer = ^Node;
  Node = Record
            Info : Kieuphantu;
            Link : pointer;
  End;
Var S : pointer;
```

Trong cách cài đặt này, ngăn xếp rỗng khi S = NIL. Ta giả sử việc cấp phát bộ nhớ động cho các phần tử mới luôn thực hiện. Do đó, ngăn xếp không bao giờ đầy và việc thêm phần tử vào stack luôn thực hiện được.

Sau đây, là các phép xử lý cơ bản trên stack tương ứng với hai cách biểu diễn trên

4.1.1. Khởi tạo ngăn xếp rỗng

Cách cài đặt bằng mảng:

```
Procedure Create-Empty(var S : Stack);
        begin
                T := 0;
        end;
```

Cách cài đặt bằng danh sách liên kết đơn:

```
Procedure Create-Empty(var S : pointer);
        begin
                T := nil;
        end;
```

4.1.2. Kiểm tra ngăn xếp rỗng

Cách cài đặt bằng mảng:

```
Function IsEmpty ( S : Stack) : Boolean;
    begin
        IsEmpty := S[T] = 0;
    end;
```

Cách cài đặt bằng danh sách liên kết đơn:

```
Function IsEmpty ( S : pointer) : Boolean;
    begin
        IsEmpty := S[T] = nil;
    end;
```

Hàm IsEmpty nhận giá trị true nếu S rỗng và false nếu S không rỗng.

4.1.3. Thêm phần tử vào ngăn xếp

Cách cài đặt bằng mảng:

Để thêm phần tử X vào đỉnh của ngăn xếp S, trước hết phải kiểm tra xem S có đầy không. Nếu S đầy thì việc bổ sung không thực hiện được, ngược lại X được bổ sung vào đỉnh của S.

```
Procedure PUSH ( var S : Stack, X : Kieuphantu);
  begin
      if  T=n then begin {kiểm tra stack đầy}
                        writeln(' Stack sẽ tràn');
                        return;
                  end;
      else   begin
                  T := T + 1; {chuyển con trỏ}
                  S[T] := X; {nạp X vào stack}
             end;
  end;
```

Cách cài đặt bằng danh sách liên kết đơn:

```
Procedure PUSH ( var S : pointer, X : Kieuphantu);
    Var P : pointer;
```

```
Begin
        {tạo phần tử mới}
        New(P);
        P^.Info := X;
        P^.Link := NIL;
        {thêm phần tử vào đỉnh}
        If S = NIL Then S := P
        Else
            begin
                P^.Link := S;
                S := P;
            end;
End;
```

4.1.4. Xóa phần tử khỏi ngăn xếp

Việc loại bỏ chỉ được thực hiện nếu S không rỗng, giá trị của phần tử bị loại bỏ được gán cho biến X.

Cách cài đặt bằng mảng:

```
procedure POP (var S : Stack; var X : Kieuphantu);
  begin
      if  IsEmpty(S) then  begin {kiểm tra stack có rỗng không}
                                write ('Stack đã cạn');
                                return;
                           end;
      else   begin
                X := S[T]; {lưu lại thông tin của phần tử sẽ bị xóa bỏ}
                T := T – 1; {chuyển con trỏ}
             end;
  end;
```

Cách cài đặt bằng danh sách liên kết đơn:

```
  Procedure POP(Var S : pointer; Var X : Kieuphantu;);
```

```
Var P : Tro;
Begin
    if IsEmpty(S) then begin {kiểm tra stack có rỗng không}
                           write ('Stack đã cạn');
                           return;
                       end;
    Else begin {loại bỏ phần tử ra khỏi đỉnh}
                P := S;
                X := S^.Info;
                S := S^.Link;
                Dispose(P);
         end;
End;
```

4.2. Hàng đợi

Hàng đợi (Queue) là một kiểu danh sách đặc biệt mà phép bổ sung một phần tử vào hàng đợi được thực hiện ở một đầu, gọi là lối sau (rear) và phép loại bỏ một phần tử được thực hiện ở đầu kia, gọi là lối trước (front).

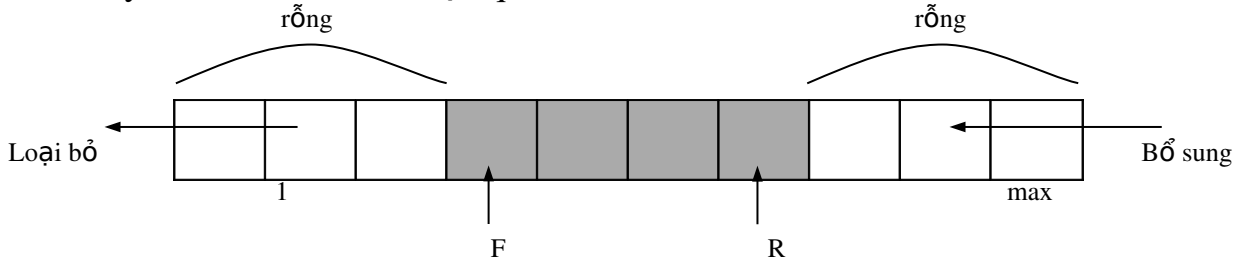
Có thể hình dung cách tổ chức lưu trữ của Queue giống như một hàng đợi: hàng người chờ mua vé tàu, học sinh xếp hàng đi vào lớp v.v...

Bởi vì, queue hoạt động theo cơ chế tự nhiên của nó là vào trước thi ra trước vào sau thì ra sau, cho nên queue còn được gọi là danh sách kiểu FIFO (First – In – First - Out).

Biểu diễn queue bằng mảng:

Tương tự như stasck chúng ta cũng có thể dùng mảng biểu diễn queue với việc sử dụng hai chỉ số F để chỉ vị trí đầu queue (lối trước) và R để chỉ vị trí cuối queue (lối sau). Khi queue rỗng thì F=R=0, nếu thêm phần tử mới vào thì R sẽ tăng lên, nếu xóa bớt phần tử thì F cũng sẽ tăng lên.

Sau đây là hình ảnh minh họa queue:



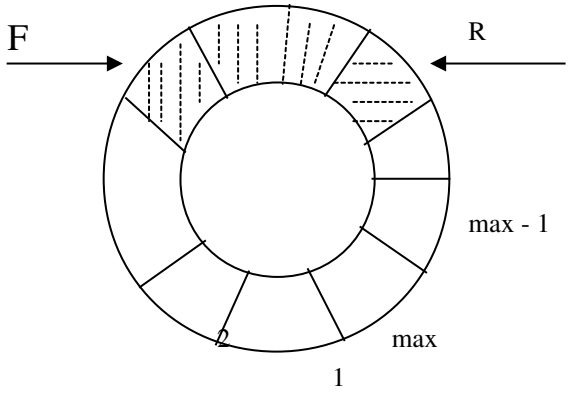
Hình 3.9: Mảng biểu diễn queue

Chúng ta khai báo cấu trúc dữ liệu biểu diễn queue như sau:

```
Const  max = N ;
Type   Queue = Record
          F,R : 0..max;
          E : Array[1..max] Of  Kieuphantu;
 End;
Var    Q : Queue;
```

Phương pháp cài đặt hàng đợi với hai chỉ số như trên có nhược điểm lớn. Nếu phép loại bỏ không thường xuyên làm cho queue rỗng, thì các chỉ số F và R sẽ tăng liên tục và sẽ vượt quá cỡ của mảng. Hàng đợi sẽ trở thành đầy, mặc dù các vị trí trống trong mảng có thể vẫn còn nhiều (do việc loại bỏ các phần tử ở đầu hàng).

Để khắc phục nhược điểm trên, chúng ta có thể xem queue như một cấu trúc vòng tròn. Tức là, Q[1] được coi như đứng sau Q[max], phần tử mới được thêm vào hàng đợi tại Q[1]. Xem hình sau:



Hình 3.10

Biểu diễn queue bằng danh sách liên kết đơn:

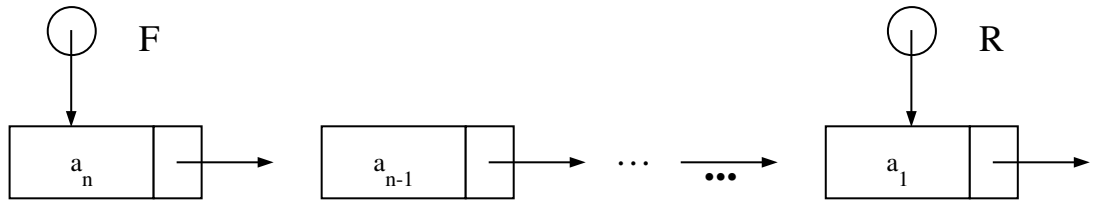
Đối với hàng đợi thì loại bỏ ở một đầu, còn bổ sung thì ở đầu kia. Nếu coi danh sách liên kết đơn như một hàng đợi thì việc loại bỏ một nút tức là loại bỏ nút đầu danh sách, còn việc bổ sung một nút tức là thêm nút mới vào cuối danh sách, nghĩa là phải tìm đến nút cuối cùng. Trong trường hợp này, để lưu trữ danh sách người ta dùng hai con trỏ, một con trỏ trỏ vào nút đầu danh sách và một con trỏ trỏ vào nút cuối danh sách.

Trong trường hợp này, chúng ta sử dụng hai con trỏ, một con trỏ F trỏ đến nút đầu hàng đợi, một con trỏ R trỏ đến nút cuối hàng đợi.

Chúng ta khai báo cấu trúc dữ liệu biểu diễn queue như sau:

```
Type    Pointer = ^Node;
  Node = Record
            Info : Kieuphantu;
            Link : pointer;
  End;
  Queue = Record
            F : pointer;
            R : pointer;
  End;
Var  Q : Queue;
```

Sau đây là hình ảnh minh họa queue:



Hình 3.11

Với cách cài đặt này, hàng đợi được xem là không khi nào đầy. Hàng đợi rỗng khi Q^.F = NIL.

Sau đây, là các phép xử lý cơ bản trên queue tương ứng với hai cách biểu diễn trên

4.2.1. Khởi tạo hàng đợi rỗng

Cách cài đặt bằng mảng:

```
Procedure Create-Empty(Var Q : Queue);
        Begin
                F := R := 0;
        End;
```

Cách cài đặt bằng danh sách liên kết đơn:

```
Procedure Create-Empty(Var Q : Queue);
        Begin
                F := R := 0;
```

```
End;
```

4.2.2. Kiểm tra hàng đợi rỗng

Cách cài đặt bằng mảng:

```
Function IsEmpty(Q : Queue) : Boolean;
    Begin
        IsEmpty := Q[F] = NIL;
    End;
```

Cách cài đặt bằng danh sách liên kết đơn:

```
Function IsEmpty(Q : Queue) : Boolean;
    Begin
        IsEmpty := Q^.F = NIL;
    End;
```

4.2.3. Thêm phần tử vào hàng đợi

Cách cài đặt bằng mảng:

Để thêm phần tử X vào hàng đợi Q, trước hết phải kiểm tra xem Q có đầy không. Nếu Q đầy thì việc bổ sung không thực hiện được, ngược lại X được bổ sung vào cuối Q.

```
Procedure AddQ(Var Q : Queue; X : Kieuphantu);
Begin
    If  F=1 and R=n or F=R+1 Then {kiểm tra hàng đợi đầy}
        write('Queue sẽ tràn')
    If F=0 Then F=R=1 {trước khi thêm vào thì hàng đợi là rỗng}
    Else  if  R=n then R =1
        Else    R := R + 1;
    Q[R] := X;
    End;
```

Cách cài đặt bằng danh sách liên kết đơn:

```
Procedure AddQ(Var Q : Queue; X : Kieuphantu);
Var P : pointer;
Begin
    {tạo một phần tử mới}
```

```
        New(P);
        P^.Info := X;
        P^.Next := NIL;
        {thêm phần tử mới vào cuối hàng đợi}
        If IsEmpty(Q) Then
                begin
                        Q^.F := P;
                        Q^.R := P;
                end
        Else
                begin
                        Q.R^.Link := P;
                        Q^.R := P;
                end;
    End;
```

4.2.4. Xóa phần tử khỏi hàng đợi

Cách cài đặt bằng mảng:

```
Procedure DeleteQ(Var Q : Queue; Var  X : Kieuphantu);
Begin
        if  F = nil then write('hang đợi trong')
        else begin
                X := Q[F];
                If  F=R  then F=R=0
                Else  if  F=n then F=1
                        Else F=R=1
                End;

End;
```

Cách cài đặt bằng danh sách liên kết đơn:

```
  Procedure Del(Var Q : Queue; Var X : Kieuphantu; Var OK : Boolean);
```

```
Var P : pointer;
Begin
    If  IsEmpty(Q) Then OK := False
    Else begin
        P := Q^.F;
        X := Q.F^.Info;
        Q^.R := Q.F^.Link;
        OK := True;
        Dispose(P);
        end;
End;
```

Bài tập thực hành của học viên

3.1 . Nêu ưu điểm và nhược điểm của phương pháp lưu trữ kế tiếp và lưu trữ móc nối đối với danh sách.

3.2 . Việc chèn một phần tử vào "giữa" danh sách (không phải là trước nút đầu và sau nút cuối) khi lưu trữ kế tiếp có gì khác so với khi lưu trữ móc nối ?

3.3 . Với danh sách nối đơn , việc truy cập vào phần tử nào được thực hiện trực tiếp ?

3.4 . Để xác định được một nút "đứng trước" một nút đã cho trong danh sách nối đơn thì phải biết những địa chỉ nào và phải làm thế nào ? Với danh sách nối vòng, có như vậy không ?

3.5 . Cho một danh sách nối đơn , có con trỏ LIST trỏ tới nút đầu tiên của danh sách này. Viết giải thuật thực hiện :

a. Cộng thêm một số A vào số đang chứa tại trường INFO của mỗi nút .

b. Đếm số lượng các nút có trong danh sách đó.

3.6. Viết chương trình con dùng cài đặt danh sách bằng mảng:

a. Nhận 1 dãy các số nguyên nhập từ bàn phím và lưu trữ nó theo đúng thứ tự nhập vào

b. Nhận 1 dãy các số nguyên nhập từ bàn phím và lưu trữ theo thứ tự ngược lại với thứ tự nhập vào.

c. In ra màn hình các phần tử trong danh sách theo thứ tự của nó trong danh sách.

3.7. Viết chương trình con như câu 3.6 nhưng dùng cài đặt danh sách bằng danh sách lien kết đơn.

3.8. Cài đặt Stack bằng cách dùng con trỏ, dùng Stack viết chương trình con đổi số nguyên dương n ở hệ thập phân sang hệ nhị phân.

3.9. Cho một Queue được lưu trữ trong bộ nhớ bởi một vectơ Q có 7 phần tử(ô nhớ) hoạt động theo cấu trúc vòng. Ban đầu Q có dạng :

Minh họa tình trạng của Q và nêu rỏ giá trị tương ứng của F và R sau mỗi lần thực hiện các phép dưới đây :

a. Thương và Hải được bổ sung vào Q

b. Loại Hùng và An ra khỏi Q

c. Kha được bổ sung vào Q

d. Loại 3 tên ra khỏi Q

Gợi ý làm bài

YÊU CẦU VỀ ĐÁNH GIÁ KẾT QUẢ HỌC TẬP:

Tiêu chí đánh giá	**Kết quả thực hiện**	**Hệ số**	**Kết qủa học tập**
Kiến thức		0,3	
Kỹ năng		0,5	
Thái độ		0,2	
Cộng:			

CHƯƠNG 4: CÁC PHƯƠNG PHÁP SẮP XẾP CƠ BẢN

Mã chương: Mh17-04

Giới thiệu:

Do các hệ thống thông tin thường phải lưu trữ một khối lượng dữ liệu đáng kể, nên việc xây dựng các giải thuật cho phép tìm kiếm nhanh sẽ có ý nghĩa rất lớn. Nếu dữ liệu trong hệ thống đã được tổ chức theo một trật tự nào đó, thì việc tìm kiếm sẽ tiến hành nhanh chóng và hiệu quả hơn. Do đó, khi xây dựng một hệ quản lý thông tin trên máy tính, bên cạnh các thuật toán tìm kiếm, các thuật toán sắp xếp dữ liệu cũng là một trong những chủ đề được quan tâm hàng đầu.

Mục tiêu:

- Trình bày được khái niệm bài toán sắp xếp;

- Mô phỏng được giải thuật, cách cài đặt, cách đánh giá giải thuật của một số phương pháp sắp xếp cơ bản;

- Giải được các bài toán sắp xếp sử dụng các phương pháp sắp xếp đã khảo sát.

- Thực hiện các thao tác an toàn với máy tính.

Nội dung chính:

1.Định nghĩa bài toán sắp xếp

Sắp xếp là một quá trình xử lý bố trí lại một danh sách các đối tượng theo thứ tự nào đó.

Ví dụ ta cần sắp xếp danh sách thí sinh theo tên với thứ tự Alphabet, hoặc sắp xếp danh sách sinh viên theo điểm trung bình với thứ tự từ cao đến thấp.

Nhìn chung có rất nhiều xử lý dữ liệu cần đến việc sắp xếp dữ liệu theo một trật tự nào đó. Ví dụ như khi cần tìm kiếm một đối tượng trong một danh sách các đối tượng bằng giải thuật tìm kiếm nhị phân thì danh sách các đối tượng này phải được sắp xếp trước đó.

Các đối tượng cần được sắp xếp thường có nhiều thuộc tính chúng ta cần chọn một thuộc tính làm khóa và sắp xếp theo khóa này.

2. Phương pháp chọn (Selection sort)

2.1.Giới thiệu phương pháp

Ta thấy rằng, nếu danh sách có thứ tự, phần tử a_i luôn là min(a_i, a_{i+1}, ., a_{n-1}). Ý tưởng của thuật toán chọn trực tiếp mô phỏng một trong những cách sắp xếp tự nhiên nhất trong thực tế: chọn phần tử nhỏ nhất trong N phần tử ban đầu, đưa phần tử này về vị trí đúng là đầu dãy hiện hành; sau đó, xem dãy hiện hành chỉ còn N-1 phần tử của dãy ban đầu, bắt đầu từ vị trí thứ 2; lặp lại quá trình trên cho dãy hiện hành... đến khi dãy hiện hành chỉ còn 1 phần tử.

2.2.Giải thuật

Đây là phương pháp sắp xếp đơn giản nhất được thực hiện như sau:

Bước 1: i = 1;

Bước 2: Tìm phần tử a[min] nhỏ nhất trong dãy hiện hành từ a[i] đến a[N]

Bước 3 : Hoán vị a[min] và a[i]

Bước 4 : Nếu i ≤ N-1 thì i = i+1; Lặp lại Bước 2
Ngược lại: Dừng. //N-1 phần tử đã nằm đúng vị trí.

Cài đặt thuật toán:

```
Procedure SelectionSort;
  Var pos,min,i,j:Integer;
    Begin
      for i:=1 to n-1 do
              Begin
                 min:=a[i]; pos:=i; {lưu lại vị trí phần tử nhỏ nhất}
                  for j:=i to n do
                    Begin
                      if(a[j]<min) then
                        Begin
                          min:=a[j]; pos:=j;
                        End;
                     End; {of for j}
```

```
            DoiCho(i,Pos) ;
        End;{of for i}
    End;
```

2.3.Ví dụ minh họa

Sắp xếp dãy gồm 10 mẩu tin có khóa là các số nguyên: 5, 6, 2, 2, 10, 12, 9, 10, 9 và 3

Bước 1: Ta chọn được phần tử có khoá nhỏ nhất (bằng 2) trong các phần tử từ a[1] đến a[10] là a[3], hoán đổi a[1] và a[3] cho nhau. Sau bước này thì a[1] có khoá nhỏ nhất là 2.

Bước 2: Ta chọn được phần tử có khoá nhỏ nhất (bằng 2) trong các phần tử từ a[2] đến a[10] là a[4], hoán đổi a[2] và a[4] cho nhau.

Tiếp tục quá trình này và sau 9 bước thì kết thúc.

Bảng sau ghi lại các giá trị khoá tương ứng với từng bước.

	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]
Ban đầu	5	6	2	2	10	12	9	10	9	3
Bước 1	**2**	6	5	2	10	12	9	10	9	3
Bước 2		**2**	5	6	10	12	9	10	9	3
Bước 3			**3**	6	10	12	9	10	9	5
Bước 4				**5**	10	12	9	10	9	6
Bước 5					**6**	12	9	10	9	10
Bước 6						**9**	12	10	9	10
Bước 7							**9**	10	12	10
Bước 8								**10**	12	10
Bước 9									**10**	12
Kết quả	**2**	**2**	**3**	**5**	**6**	**9**	**9**	**10**	**10**	**12**

3. Phương pháp chèn (Insertion sort)

3.1.Giới thiệu phương pháp

Giả sử có một dãy $\mathbf{a_1, a_2, ..., a_n}$ trong đó i phần tử đầu tiên $\mathbf{a_1, a_2, ..., a_{i-1}}$ đã có thứ tự. Ý tưởng chính của giải thuật sắp xếp bằng phương pháp chèn trực tiếp là tìm cách chèn phần tử $\mathbf{a_i}$ vào vị trí thích hợp của đoạn đã được sắp để có dãy mới $\mathbf{a_1, a_2, ..., a_i}$ trở nên có thứ tự. Vị trí này chính là vị trí giữa hai phần tử $\mathbf{a_{k-1}}$ và $\mathbf{a_k}$ thỏa $\mathbf{a_{k-1} \le a_i < a_k}$ $(1 \le k \le i)$.

Cho dãy ban đầu $\mathbf{a_1, a_2, ..., a_n}$, ta có thể xem như đã có đoạn gồm một phần tử $\mathbf{a_1}$ đã được sắp, sau đó thêm $\mathbf{a_2}$ vào đoạn $\mathbf{a_1}$ sẽ có đoạn $\mathbf{a_1\, a_2}$ được

sắp; tiếp tục thêm $\mathbf{a_3}$ vào đoạn $\mathbf{a_1}\ \mathbf{a_2}$ để có đoạn $\mathbf{a_1}\ \mathbf{a_2}\ \mathbf{a_3}$ được sắp; tiếp tục cho đến khi thêm xong $\mathbf{a_N}$ vào đoạn $\mathbf{a_1}\ \mathbf{a_2}\ldots\mathbf{a_{N-1}}$ sẽ có dãy $\mathbf{a_1}\ \mathbf{a_2}\ldots\ \mathbf{a_N}$ được sắp

3.2.Giải thuật

Các bước thực hiện ý tưởng giải thuật như sau:

Bước 1: i = 2; // giả sử có đoạn a[1] đã được sắp

Bước 2: x = a[i];

Tìm vị trí j thích hợp trong đoạn a[1] đến a[i-1] để chèn a[i] vào

Bước 3: Dời chỗ các phần tử từ a[j] đến a[i-1] sang phải 1 vị trí để dành chổ cho a[i]

Bước 4: a[j] = x; // có đoạn a[1]..a[i] đã được sắp

Bước 5: i = i+1;

Nếu i ≤ n : Lặp lại Bước 2.

Ngược lại : Dừng.

Cài đặt thuật toán:

```
Procedure InsSort;
Var i,j :Integer ;
Begin
   for i:=2 to n do  {xem như phần tử đầu tiên đã sắp xếp}
        Begin
          Tam:=A[i];  {Lưu lại phần tử đang xét để tìm vị trí chèn vào}
          j:=i-1;     { duyệt dãy con còn lại bắt đầu từ i-1}
          while(Tam<A[j]) do
                    Begin
                          A[j+1]:=A[j];{nhích các phần tử ra 1 vị trí}
                          j:=j-1;
                     End;
          A[j+1]:=Tam;
        End; {for}
End;
```

3.3.Ví dụ minh họa

Sắp xếp dãy gồm 10 mẩu tin đã cho ở trên có khóa là các số nguyên: 5, 6, 2, 2, 10, 12, 9, 10, 9 và 3.

Bước 1: Xen a[2] vào dãy chỉ có một phần tử a[1] ta được dãy hai phần tử a[1]..a[2] có thứ tự. Việc xen này thực ra không phải làm gì cả vì hai phần tử a[1], a[2] có khoá tương ứng là 5 và 6 đã có thứ tự.

Bước 2: Xen a[3] vào dãy a[1]..a[2] ta được dãy ba phần tử a[1]..a[3] có thứ tự. Việc xen này được thực hiện bằng cách : so sánh khoá của a[3] với khoá của a[2], do khoá của a[3] nhỏ hơn khoá của a[2] (2<6) nên hoán đổi a[3] và a[2] cho nhau. Lại so sánh khoá của a[2] với khoá của a[1], do khoá của a[2] nhỏ hơn khoá của a[1] (2<5) nên hoán đổi a[2] và a[1] cho nhau.

Tiếp tục quá trình này và sau 9 bước thì kết thúc.

Bảng sau ghi lại các giá trị khoá tương ứng với từng bước.

	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]
Ban đầu	5	6	2	2	10	12	9	10	9	3
Bước 1	5	6								
Bước 2	2	5	6							
Bước 3	2	2	5	6						
Bước 4	2	2	5	6	10					
Bước 5	2	2	5	6	10	12				
Bước 6	2	2	5	6	9	10	12			
Bước 7	2	2	5	6	9	10	10	12		
Bước 8	2	2	5	6	9	9	10	10	12	
Bước 9	2	2	3	5	6	9	9	10	10	12

4. Phương pháp đổi chỗ (Interchange sort)

4.1.Giới thiệu phương pháp

Ý tưởng chính của giải thuật là xuất phát từ đầu dãy, tìm tất cả thứ tự ngược chứa phần tử này, triệt tiêu chúng bằng cách đổi chỗ phần tử này với phần tử tương ứng trong cặp thứ tự ngược. Lặp lại xử lý trên với các phần tử tiếp theo trong dãy.

4.2.Giải thuật

Các bước thực hiện ý tưởng giải thuật:

Bước 1 : i = 1;// bắt đầu từ đầu dãy

Bước 2 : j = i+1;//tìm các phần tử a[j] < a[i], j>i

Bước 3 :

Trong khi j ≤ N thực hiện

Nếu a[j]<a[i]: a[i]<->a[j]; //xét cặp a[i], a[j]

j = j+1;

Bước 4 : i = i+1;

Nếu i< n: Lặp lại Bước 2.

Ngược lại: Dừng.

Cài đặt thuật toán:

```
Procedure InterchangeSort;
Var    i, j: integer;
Begin
        for  i: = 1 to n  do
                for  j: =i+1 to  n  do
                        if (a[j]< a[i]) { nếu có sự sai vị trí thì đổi chỗ}
                                Hoanvi(a[i],a[j]);
End;
```

4.3.Ví dụ minh họa

Sắp xếp dãy gồm 10 mẩu tin có khóa là các số nguyên: 5, 6, 2, 2, 10, 12, 9, 10, 9 và 3

Bảng sau ghi lại các giá trị khoá tương ứng với từng bước.

	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]
Ban đầu	5	6	2	2	10	12	9	10	9	3
i= 1	2	6	5	2	10	12	9	10	9	3
i= 2	2	5	6	2	10	12	9	10	9	3
	2	2	6	5	10	12	9	10	9	3
i= 3	2	2	5	6	10	12	9	10	9	3
	2	2	3	6	10	12	9	10	9	5
i= 4	2	2	3	5	10	12	9	10	9	6
i= 5	2	2	3	5	9	12	10	10	9	6
	2	2	3	5	6	12	10	10	9	9
i= 6	2	2	3	5	6	10	12	10	9	9
	2	2	3	5	6	9	12	10	10	9
i= 7	2	2	3	5	6	9	10	12	10	9
	2	2	3	5	6	9	9	12	10	10
i= 8	2	2	3	5	6	9	9	10	12	10

i= 9	2	2	3	5	6	9	9	10	10	12

5.Phương pháp nổi bọt (Bubble sort)

5.1.Giới thiệu phương pháp

Ý tưởng chính của giải thuật là xuất phát từ cuối (đầu) dãy, đổi chỗ các cặp phần tử kế cận để đưa phần tử nhỏ (lớn) hơn trong cặp phần tử đó về vị trí đúng đầu (cuối) dãy hiện hành, sau đó sẽ không xét đến nó ở bước tiếp theo, do vậy ở lần xử lý thứ i sẽ có vị trí đầu dãy là i . Lặp lại xử lý trên cho đến khi không còn cặp phần tử nào để xét.

5.2.Giải thuật

Các bước thực hiện ý tưởng giải thuật:

Bước 1 : i = 1; // lần xử lý đầu tiên

Bước 2 : j = N; //Duyệt từ cuối dãy ngược về vị trí i

Trong khi (j > i) thực hiện:

Nếu a[j]<a[j-1]: a[j]<->a[j-1];//xét cặp phần tử kế cận

j = j-1;

Bước 3 : i = i+1; // lần xử lý kế tiếp

Nếu i >N-1: Hết dãy. Dừng

Ngược lại : Lặp lại Bước 2

Cài đặt thuật toán:

```
Procedure BubbleSort;
Var i,j :Integer ;
Begin
        for i:=1 to n-1 do
                for j:=n downto i+1 do
                        if(a[j]<a[j-1]) then
                                DoiCho(j,j-1) ;
End;
```

5.3.Ví dụ minh họa

Sắp xếp dãy gồm 10 mẩu tin đã cho ở trên có khóa là các số nguyên: 5, 6, 2, 2, 10, 12, 9, 10, 9 và 3.Bước 1: Xét a[10] có khoá là 3, nhỏ hơn khoá của a[9] nên ta hoán đổi a[10] và a[9] cho nhau. Khoá của a[9] bây giờ là 3 nhỏ hơn khoá của a[8] nên ta hoán đổi a[9] và a[8] cho nhau. Khoá của a[8] bây giờ là 3 nhỏ hơn khoá của a[7] nên ta hoán đổi a[8] và a[7] cho nhau. Khoá của a[7] bây giờ là 3 nhỏ hơn khoá của a[6] nên ta hoán đổi a[7] và a[6] cho nhau. Khoá của a[6] bây giờ là 3 nhỏ hơn khoá của a[5] nên ta hoán đổi a[6] và a[5] cho nhau. Khoá của a[5] bây giờ là 3 **không nhỏ hơn** khoá của a[4] nên bỏ qua. Khoá của a[4] là 2 **không nhỏ hơn** khoá của a[3] nên bỏ qua. Khoá của a[3] là 2 nhỏ hơn khoá của a[2] nên ta hoán đổi a[3] và a[2] cho nhau. Khoá của a[2] bây giờ là 2 nhỏ hơn khoá của a[1] nên ta hoán đổi a[2] và a[1] cho nhau. Đến đây kết thúc bước 1 và a[1] có khoá nhỏ nhất là 2.

Bước 2: Xét a[10] có khoá là 9, nhỏ hơn khoá của a[9] nên ta hoán đổi a[10] và a[9] cho nhau. Khoá của a[9] bây giờ là 9 **không nhỏ hơn** khoá của a[8] nên bỏ qua. Khoá của a[8] là 9 nhỏ hơn khoá của a[7] nên ta hoán đổi a[8] và a[7] cho nhau. Khoá của a[7] bây giờ là 9 nhỏ hơn khoá của a[6] nên ta hoán đổi a[7] và a[6] cho nhau. Khoá của a[6] bây giờ là 9 **không nhỏ hơn** khoá của a[5] nên bỏ qua. Khoá của a[5] bây giờ là 3 **không nhỏ hơn** khoá của a[4] nên bỏ qua. Khoá của a[4] là 2 nhỏ hơn khoá của a[3] nên ta hoán đổi a[4] và a[3] cho nhau. Khoá của a[3] bây giờ là 2 nhỏ hơn khoá của a[2] nên ta hoán đổi a[3] và a[2] cho nhau. Đến đây kết thúc bước 2 và a[2] có khoá là 2.

Tiếp tục quá trình này và sau 9 bước thì kết thúc.

Bảng sau ghi lại các giá trị khoá tương ứng với từng bước.

	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]
Ban đầu	5	6	2	2	10	12	9	10	9	3
Bước 1	2	5	6	2	3	10	12	9	10	9
Bước 2		2	5	6	3	9	10	12	9	10
Bước 3			3	5	6	9	9	10	12	10
Bước 4				5	6	9	9	10	10	12
Bước 5					6	9	9	10	10	12
Bước 6						9	9	10	10	12
Bước 7							9	10	10	12
Bước 8								10	10	12
Bước 9									10	12
Kết quả	**2**	**2**	**3**	**5**	**6**	**9**	**9**	**10**	**10**	**12**

6.Phương pháp sắp xếp nhanh (Quick sort)

6.1.Giới thiệu phương pháp

Phương pháp này khác hẳn với các phương pháp mà ta đã xét, đây là phương pháp có hiệu lực cao về thời gian thực hiện trung bình, dùng phương pháp 'Chia để trị'. Trong thuật toán này, phần tử được chọn là bất kỳ mà ta gọi là chốt.

Khi một phần tử của dãy A đã được chọn làm "chốt" thì mọi phần tử nhỏ hơn chốt sẽ được chuyển về phía trước chốt, mọi phần tử lớn hơn chốt sẽ được chuyển về phía sau chốt. Cuối cùng các phần tử nhỏ hơn chốt sẽ tạo thành một dãy con thứ nhất, các phần tử lớn hơn chốt chốt sẽ tạo thành dãy con thứ hai. Vị trí nằm giữa hai dãy con này chính là vị trí của chốt trong sắp xếp.

Như vậy dãy A đã được phân làm hai dãy con. Đối với từng dãy con này một chiến thuật tương tự sẽ được áp dụng, và cứ như vậy cho đến khi mảng con chỉ còn là một phần tử.

6.2.Giải thuật

Các bước thực hiện ý tưởng giải thuật:

Để sắp xếp dãy a[l], a[$_1$, ., a_r ta tiến hành các bước sau:

Bước 1: Xác định chốt

Bước 2: Phân chia dãy a_l, a_{l+1}, ., a_r thành 2 dãy con a[l].. a[k-1], a[k].. a[r]

Bước 3: Sắp xếp dãy a[l].. a[k-1] *{đệ quy}*

Bước 4: Sắp xếp dãy a[k].. a[r] *{đệ quy}*

Giải thuật phân hoạch dãy a_l, a_{l+1}, ., a_r thành 2 dãy con:

Bước 1 : Chọn tùy ý một phần tử a[k] trong dãy là giá trị chốt, $l \le k \le r$:

x = a[k]; i = l; j = r;

Bước 2 : Phát hiện và hiệu chỉnh cặp phần tử a[i], a[j] nằm sai chỗ :

+ Bước 2a : Trong khi (a[i]<x) i++;

+ Bước 2b : Trong khi (a[j]>x) j--;

+ Bước 2c : Nếu i< j // a[i] ≥ x ≥ a[j] mà a[j] đứng sau a[i]

Hoán vị (a[i],a[j]);

Bước 3 :

Nếu i < j: Lặp lại Bước 2.//chưa xét hết mảng

Nếu i ≥ j: Dừng

Cài đặt thuật toán:

```
Procedure Q_Sort( L,R:Integer);
Var i,j,chot:Integer;
  Begin
     if(L<R) Then
     Begin
       chot:=A[L]; i:=L; j:=R+1;
        Repeat
          i:=i+1;
          While(A[i]<chot) and (i<=R) Do i:=i+1;
          j:=j-1;
          While(A[j]>chot)   Do j:=j-1;
          if(i<j)  then DoiCho(i,j);
        Until (i>=j);
        DoiCho(j,L);
     End
     else Exit;
  Q_Sort(L,j-1);
  Q_Sort(j+1,R);
  End;
```

6.3.Ví dụ minh họa

Sắp xếp dãy gồm 10 mẩu tin có khóa là các số nguyên: 5, 6, 2, 2, 10, 12, 9, 10, 9 và 3

Phân chia đoạn l=1, r=10, chọn giá trị chốt x=A[5]=10

	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]
Ban đầu	5	6	2	2	10	12	9	10	9	3
Bước 1	5	6	2	2	3	12	9	10	9	10
Bước 2	5	6	2	2	3	10	9	10	9	12

Bước 3	5	6	2	2	3	9	9	10	10	12
Bước 4	5	6	2	2	3	9	9	10	10	12

Sau các bước tìm và đổi chổ trên chúng ta có tất cả các số nhỏ hơn chốt x=10 đều ở phái trước x, các số lơn hơn chốt x=10 đều ở phía sau x. Dãy A đã được phân thành 2 dãy con:

- Dãy con thứ 1 : 5, 6, 2, 2, 3, 9, 9

- Dãy con thứ 2: 10, 12

- Còn giá trị chốt x=10 đã đúng vị trí sắp xếp của nó trong dãy A

Quá trình sắp xếp lại được tiếp tục với từng dãy con, bằng một kỹ thuật tương tự:

Phân đoạn: Dãy con 1: l=1, r=7, x=A[3]=2 **Dãy con 2: l=9, r=10, x=A[9]=10**

	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]
Ban đầu	5	6	2	2	3	9	9	10	10	12
Bước 1	2	6	2	5	3	9	9	10	10	12
Bước 2	2	2	6	5	3	9	9	10	10	12

Sau các bước tìm và đổi chổ trên chúng ta có dãy con thứ 2 đã được sắp xếp đúng thứ tự, dãy con thứ 1 được phân thành 2 dãy con:

- Dãy con thứ 3: chỉ có 1 phần tử bằng 2 xem như đã được sắp xếp đúng vị trí

- Dãy con thứ 4: 6, 5, 3, 9, 9

- Giá trị chốt x=2 đã đúng vị trí sắp xếp của nó

Phân đoạn: Dãy con 4: l=3, r=7, x=A[5]=3

	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]
Ban đầu	2	2	6	5	3	9	9	10	10	12
Bước 1	2	2	3	5	6	9	9	10	10	12

…										
Kết quả	**2**	**2**	**3**	**5**	**6**	**9**	**9**	**10**	**10**	**12**

Bài tập thực hành của học viên

4.1. Cho dãy số: 3 7 4 5 17 2 6 9

a. Minh hoạ việc sắp xếp dãy số trên theo chiều tăng dần bằng 5 phương pháp đã học.

4.2. Cho dãy số ở câu 4.1

a. Minh hoạ việc sắp xếp dãy số trên theo chiều giảm dần bằng 5 phương pháp đã học.

b. Viết thủ tục sắp xếp dãy theo các phương pháp nói trên.

YÊU CẦU VỀ ĐÁNH GIÁ KẾT QUẢ HỌC TẬP:

Tiêu chí đánh giá	**Kết quả thực hiện**	**Hệ số**	**Kết qủa học tập**
Kiến thức		0,3	
Kỹ năng		0,5	
Thái độ		0,2	
Cộng:			

CHƯƠNG 5: TÌM KIẾM

Mã chương: Mh17-05

Giới thiệu:

Trong hầu hết các hệ lưu trữ, quản lý dữ liệu, thao tác tìm kiếm thường được thực hiện nhiều nhất để khai thác thông tin. Các thuật toán sắp xếp và tìm kiếm cùng với các kỹ thuật được sử dụng trong đó được coi là các kỹ thuật cơ sở cho lập trình máy tính.

Mục tiêu:

- Trình bày được giải thuật, cài đặt được giải thuật và đánh giá được độ phức tạp của giải thuật tìm kiếm tuyến tính, tìm kiếm nhị phân.

- Giải được các bài toán sử dụng giải thuật tìm kiếm tuyến tính, tìm kiếm nhị phân.

- Thực hiện các thao tác an toàn với máy tính.

Nội dung chính:

Tìm kiếm một phần tử nào đó của một tập đối tượng theo một tiêu chí đề ra là bài toán phổ biến trong tin học.

Ở đây ta xét tới một dạng đơn giản của nó:

"cho một vec tơ A bao gồm n phần tử, có giá trị là các số khác nhau : A[1], A[2], A[3],…, A[n]"

"cho một số X, hãy tìm xem có phần tử nào của A mà giá trị của nó bằng X không"

Tìm kiếm sẽ "được thỏa" khi có, hoặc "không thỏa" khi không có phần tử nào có giá trị bằng X

1.Tìm kiếm tuyến tính

1.1.Giới thiệu phương pháp

Tìm tuyến tính là một kỹ thuật tìm kiếm rất đơn giản và cổ điển. Thuật toán tiến hành so sánh **x** lần lượt với phần tử thứ nhất, thứ hai, ... của dãy khóa cho đến khi gặp được phần tử có khóa cần tìm, hoặc đã tìm hết mảng mà không thấy **x**.

1.2.Giải thuật

Các bước thực hiện ý tưởng giải thuật:

Bước 1: i = 1; // bắt đầu từ phần tử đầu tiên của dãy

Bước 2: So sánh a[i] với x, có 2 khả năng :

+ a[i] = x : Tìm thấy. Dừng

+ a[i] != x : Sang Bước 3.

Bước 3 : i = i+1; // xét tiếp phần tử kế trong mảng

Nếu i >N: Hết mảng,không tìm thấy.Dừng

Ngược lại: Lặp lại Bước 2.

Cài đặt thuật toán:

```
Function TKTT(A,n,X):Integer;
Begin
        i:=1;  A[n+1]:=X;
       While  A[i] < > X do i:=i+1;
          {Tìm thấy hay không}
        if  i=n+1 then  TKTT:=0
        else          TKTT:=i;
End;
```

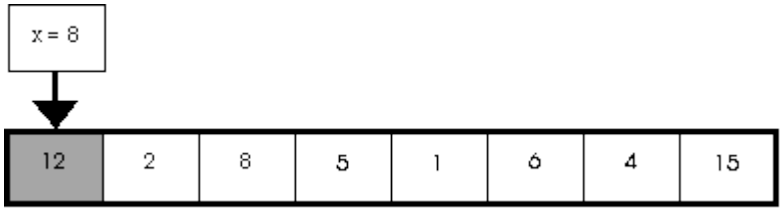
1.3.Ví dụ minh họa

Cho dãy số a:

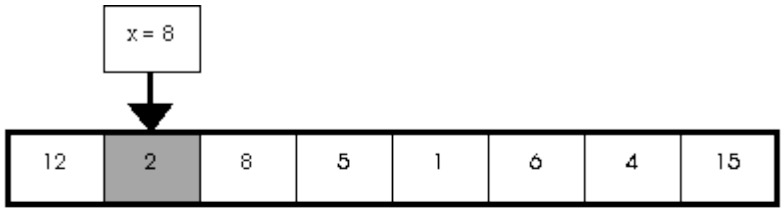
12 2 8 5 1 6 4 15

Nếu giá trị cần tìm là 8, giải thuật được tiến hành như sau :

i = 1



i = 2



i = 3

Dừng.

2.Tìm kiếm nhị phân

2.1.Giới thiệu phương pháp

Nếu các phần tử của A đã được sắp xếp, thì tìm kiếm nhị phân là phương pháp tìm kiếm khá thông dụng. Nó tương tự như cách thức ta hay làm như tra một từ trong từ điển. Đối với phép tìm kiếm nhị phân thì ta luôn chọn khoá ở giữa, giả sử dẫy đang xét là A_1, …,A_2 thì khoá ở giữa dãy sẽ là A_i với i= (l+r) div 2. Nếu X<A_i thì tìm kiếm được thực hiện tiếp với A_1,…,A_{i-1}; còn nếu ngược lại tìm kiếm lại được làm với A_{i+1},…,A_n

2.2.Giải thuật

Các bước thực hiện ý tưởng giải thuật:

Bước 1: left = 1; right = N; // tìm kiếm trên tất cả các phần tử

Bước 2:

mid = (left+right)/2; // lấy mốc so sánh

So sánh a[mid] với x, có 3 khả năng :

+ a[mid] = x: Tìm thấy. Dừng

+ a[mid] > x: //tìm tiếp x trong dãy con a_{left} .. $a_{mid\ -1}$:

right =midle - 1;

+ a[mid] < x: //tìm tiếp x trong dãy con amid +1 .. aright :

left = mid + 1;

Bước 3:

Nếu left ≤ right //còn phần tử chưa xét -> tìm tiếp.

Lặp lại Bước 2.

Ngược lại: Dừng; //Đã xét hết tất cả các phần tử.

Cài đặt thuật toán:

```
Function B_search(A,n,X)
Var l,r,m:Integer;
Begin
        l:=1;  r:=n;
         While  l<=r do
               Begin
                     m:=(l+r)div 2;
                    if  X < A[m] then  r:=m-1;
                    else
                          if  X > A[m] then l:=m+1;
                          else
                                Begin
                                      B_Search:=m; Exit;
                                End;
               end;
          B_Search:=0;
        End;
```

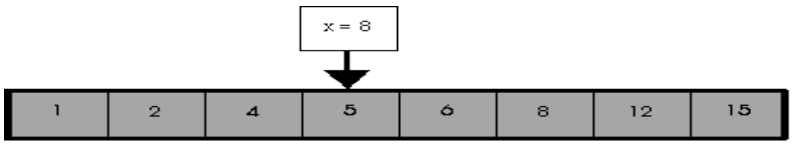
2.3.Ví dụ minh họa

Cho dãy số a gồm 8 phần tử:

1 2 4 5 6 8 12 15

Nếu giá trị cần tìm là 8, giải thuật được tiến hành như sau:

left = 1, right = 8, midle = 4



left = 5, right = 8, midle = 6

Dừng.

Bài tập thực hành của học viên

5.1. Cho dãy số : -10 8 9 6 12 7. Minh họa việc tìm kiếm số k=15 trong dãy trên theo phương pháp tìm kiếm tuần tự.

5.2. So với tìm kiếm tuần tự thì tìm kiếm nhị phân có ưu điểm gì? Nhược điểm gì?

5.3. Cho danh sách các tên sinh viên được xếp theo thứ tự sau:

1	2	3	4	5	6	7	8	9	10
An	Binh	Cúc	Giao	Hùng	Kiên	Lộc	Ninh	Sơn	Vy

Áp dụng thuật toán tìm kiếm nhị phân(Binary Search) để tìm tên Cúc, Sơn trong danh sách. Nêu rỏ giá trị của biến L, R, Mid ứng với từng lược

YÊU CẦU VỀ ĐÁNH GIÁ KẾT QUẢ HỌC TẬP:

Tiêu chí đánh giá	**Kết quả thực hiện**	**Hệ số**	**Kết qủa học tập**
Kiến thức		0,3	
Kỹ năng		0,5	
Thái độ		0,2	
Cộng:			

CHƯƠNG 6: CÂY

Mã chương: Mh17-06

Giới thiệu:

Cây là một cấu trúc rất gần gũi và có nhiều ứng dụng trong thực tế. Cây là một cấu trúc phân cấp trên một tập hợp nào đó các đối tượng. Một ví dụ quen thuộc về cây, đó là cây thư mục hoặc mục lục của cuốn sách cũng là một cây. Cây được sử dụng rộng rãi trong rất nhiều vấn đề khác nhau.

Mục tiêu:

- Trình bày được khái niệm về cây, cây nhị phân;

- Cài đặt được cây trên máy tính bằng các cấu trúc mảng và cấu trúc danh sách liên kết;

- Giải được bài toán duyệt cây nhị phân.

- Thực hiện các thao tác an toàn với máy tính.

Nội dung gồm:

1. Khái niệm về cây và cây nhị phân

1.1. Các khái niệm về cây

- Một cây là tập hợp hữu hạn các nút trong đó có một nút đặc biệt gọi là gốc (root). Giữa các nút có một quan hệ phân cấp gọi là "quan hệ cha con".

- Cây được định nghĩa đệ qui như sau:

1. Một nút là một cây và nút này cũng là gỗc của cây.

2. Giả sử T_1, T_2, …,T_n (n 1) là các cây có gốc tương ứng r_1, r_2,…, r_n. Khi đó cây T với gốc r được hình thành bằng cách cho r trở thành nút cha của các nút r_1, r_2,…, r_n

Bậc của một nút: là số con của nút đó

Bậc của một cây: là bậc lớn nhất của các nút có trên cây đó (số cây con tối đa của một nút thuộc cây). Cây có bậc n thì gọi là cây n - phân

Nút gốc: là nút có không có nút cha

Nút lá: là nút có bậc bằng 0

Nút nhánh: là nút có bậc khác 0 và không phải là nút gốc

Mức của một nút

Mức (gốc (T_0)) =1

Gọi T_1, T_2,..., T_n là các cây con của T_0.

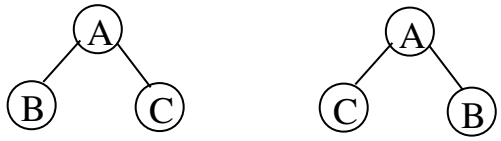
Khi đó Mức (T_1) = Mức (T_2) = ... = Mức (T_n) = Mức (T_0) +1

Chiều cao của cây: là số mức lớn nhất có trên cây đó

Đường đi: Dãy các đỉnh n_1, n_2, ...,n_k được gọi là đường đi nếu n_i là cha của n_{i+1} (1 i k-1

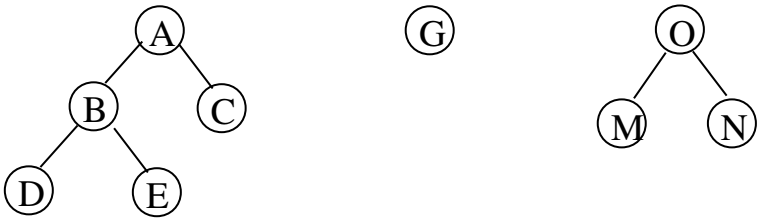
Độ dài của đường đi: là số nút trên đường đi -1

Cây được sắp : Trong một cây, nếu các cây con của mỗi đỉnh được sắp theo một thứ nhất định, thì cây được gọi là cây được sắp (cây có thứ tự). Chẳng hạn, hình minh hoạ hai cây được sắp khác nhau



Hình 6.1. Hai cây được sắp khác nhau

Rừng: là tập hợp hữu hạn các cây phân biệt

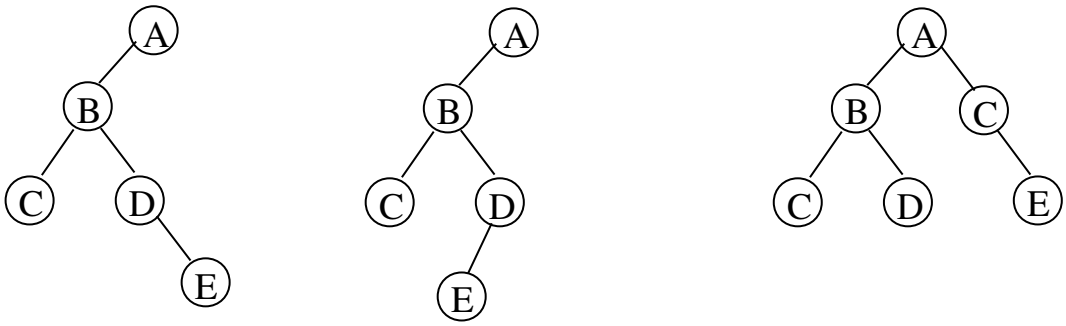


Hình 6.2. Rừng gồm ba cây

1.2. Khái niệm cây nhị phân

Cây nhị phân là cây mà mỗi nút có tối đa hai cây con. Đối với cây con của một nút người ta cũng phân biệt cây con trái và cây con phải.

Như vậy cây nhị phân là cây có thứ tự.



Hình 6.3 . Một số cây nhị phân

Đối với cây nhị phân cần chú ý tới một số tính chất sau

i) Số lượng tối đa các nút có ở mức i trên cây nhị phân là 2^{i-1} *(i 1)*

ii) Số lượng nút tối đa trên một cây nhị phân có chiều cao h là 2^h*-1(h 1)*

2. Biểu diễn cây nhị phân và cây tổng quát

2.1. Biểu diễn cây nhị phân

Lưu trữ kế tiếp

Phương pháp tự nhiên nhất để biểu diễn cây nhị phân là chỉ ra đỉnh con trái và đỉnh con phải của mỗi đỉnh.

Ta có thể sử dụng một mảng để lưu trữ các đỉnh của cây nhị phân. Mỗi đỉnh của cây được biểu diễn bởi bản ghi gồm ba trường:

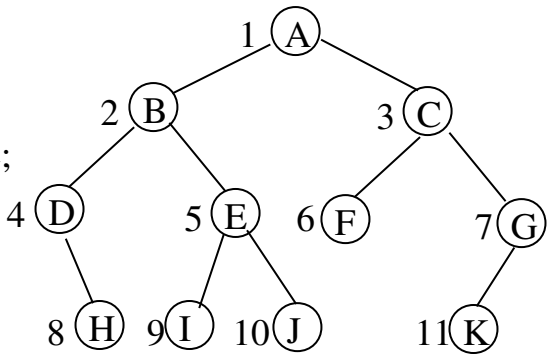
trường infor: mô tả thông tin gắn với mỗi đỉnh

letf : chỉ đỉnh con trái

right: chỉ đỉnh con phải.

Giả sử các đỉnh của cây được được đánh số từ 1 đến max. Khi đó cấu trúc dữ liệu biểu diễn cây nhị phân được khai báo như sau:

```
const  max  = ...;  {số thứ tự lớn nhất của nút trên cây}
type
        item  = ...; {kiểu dữ liệu của các nút trên cây}
        Node = record
                infor : item;
                letf :0..max;
                right :0..max;
           end;
      Tree = array[1.. max] of Node;
```



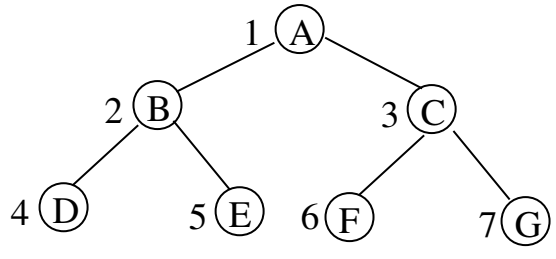
Hình 6.4. Một cây nhị phân

Hình 5.5. Minh hoạ cấu trúc dữ liệu biểu diễn cây nhị phân trong hình 5.4

	infor	left	right
1	A	2	3
2	B	4	5
3	C	6	7
4	D	0	8
5	E	9	10
6	F	0	0
7	G	11	9
8	H	0	0
9	I	0	0
0	J	0	0
1	K	0	0

Hình 6.5. Cấu trúc dữ liệu biểu diễn cây

Nếu có một cây nhị phân hoàn chỉnh đầy đủ, ta có thể dễ dàng đánh số cho các nút trên cây đó theo thứ tự lần lượt từ mức 1 trở lên, hết mức này đến mức khác và từ trái qua phải đối với các nút ở mỗi mức. Ví dụ với hình 5.6 có thể đánh số như sau:



Hình 6.6. Cây nhị phân được đánh số

Ta có nhận xét sau: con của nút thứ i là các nút thứ 2i và 2i + 1 hoặc cha của nút thứ j là j/2 .

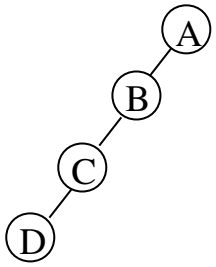
Nếu như vậy thì ta có thể lưu trữ cây này bằng một vectơ V, theo nguyên tắc: nút thứ i của cây được lưu trữ ở V[i]. Đó chính là cách lưu trữ kế tiếp đối với cây nhị phân. Với cách lưu trữ này nếu biết được địa chỉ của nút con sẽ tính được địa chỉ nút cha và ngược lại.

Như vậy với cây đầy đủ nêu trên thì hình ảnh lưu trữ sẽ như sau

A	B	C	D	E	F	G
v[1]	v[2]	v[3]	v[4]	v[5]	v[6]	v[7]

Nhận xét

Nếu cây nhị phân không đầy đủ thì cách lưu trữ này không thích hợp vì sẽ gây ra lãng phí bộ nhớ do có nhiều phần tử bỏ trống (ứng với cây con rỗng). Ta hãy xét cây như hình 5.7. Để lưu trữ cây này ta phải dùng mảng gồm 31 phần tử mà chỉ có 5 phần tử khác rỗng; hình ảnh lưu trữ miền nhớ của cây này như sau:



Hình 6.7.Cây nhị phân đặcbiệt

A	B		C				D								E		...

(: chỉ chỗ trống)

Nếu cây nhị phân luôn biến động nghĩa là có phép bổ sung, loại bỏ các nút thường xuyên tác động thì cách lưu trữ này gặp phải một số nhược điểm như tốn thời gian khi phải thực hiện các thao tác này, độ cao của cây phụ thuộc vào kích thước của mảng...

Lưu trữ móc nối

Cách lưu trữ này khắc phục được các nhược điểm của cách lưu trữ trên đồng thời phản ánh được dạng tự nhiên của cây.

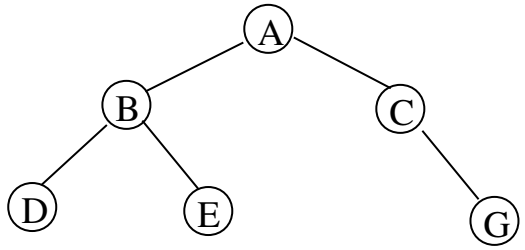
Trong cách lưu trữ này mỗi nút tương ứng với một phần tử nhớ có qui cách như sau:

letf	info	right

Trường info ứng với thông tin (dữ liệu) của nút

Trường left ứng với con trỏ, trỏ tới cây con trái của nút đó

Trường right ứng với con trỏ, trỏ tới cây con phải của nút đó

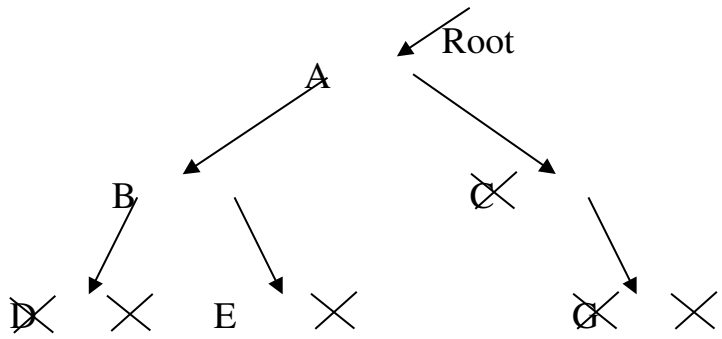


Hình 6.8

Ta có thể khai báo như sau:

```
Type
    item  = ...;{kiểu dữ liệu của các nút trên cây  }
    Tree  = ^Node;
    Node = record
        info : item;
        left, right: Tree;
        end;
var  Root :Tree;
```

Ví dụ: cây nhị phân hình 5.8 có dạng lưu trữ móc nối như ở hình 5.9



Hình 6.9. Cấu trúc dữ liệu biểu diễn cây

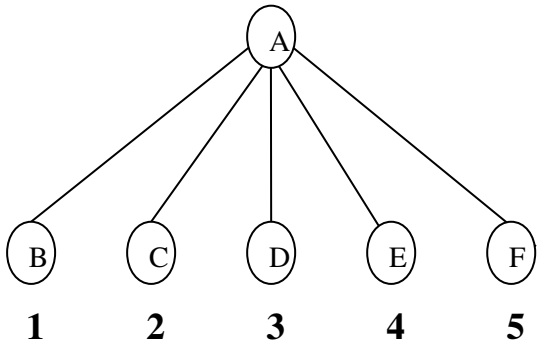
Để truy nhập vào các nút trên cây cần có một con trỏ Root, trỏ tới nút gốc của cây

2.2. Biểu diễn cây tổng quát

Biểu diễn cây tổng quát bằng một cây nhị phân gọi là cây nhị phân tương đương.

Với cách biểu diễn này chúng ta gán số thứ tự cho các cây con của mỗi nút. Giả sử trên hình vẽ ta đánh số thứ tự từ trái sang phải.

Chẳng hạn với nút có 5 cây con sau đây,thì sẽ đánh số như



HÌNH 6.10

Chúng ta xem: nút 1 là "con cả" của nút A, nút 2 là "em kề nút 1".,nút 3 là "em kề nút 2" v.v…. Với mỗi nút chúng ta chỉ cần chú ý tới hai quan hệ này là đủ

Từ đó quy cách của mỗi nút trên cây nhị phân tương đương, có dạng :

LCC	INFO	REK

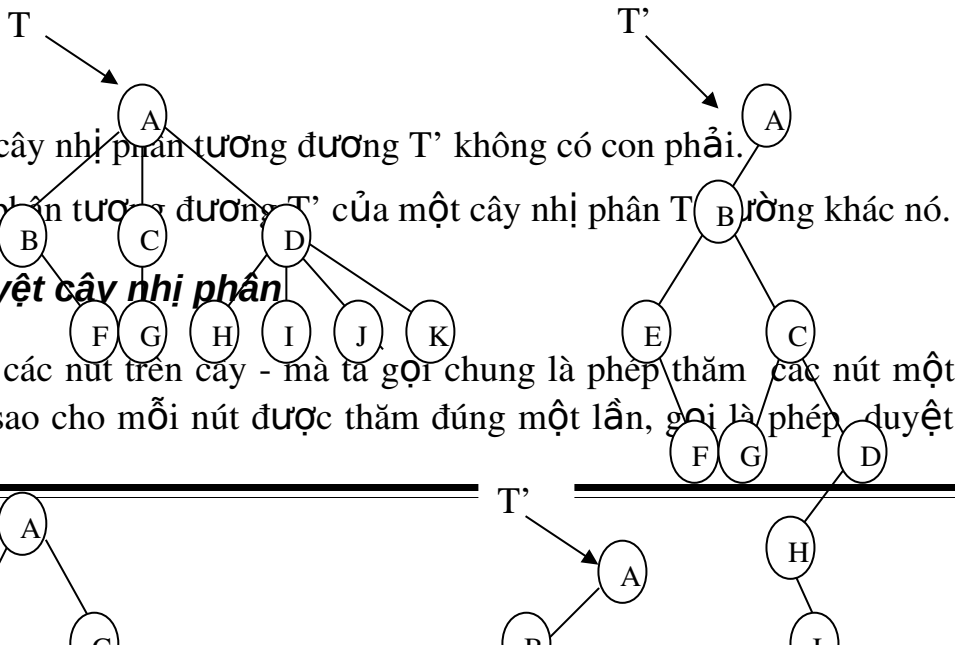
Với LCC là trường con trỏ, ở đó chứa địa chỉ của nút "con cả"

REK là trường con trỏ, ở đó chứa địa chỉ của nút "em kề nó"

Như vậy với bất kì một cây con nào cũng có một và chỉ một cây nhị phân tương đương với nó. Điều đó cũng có nghĩa là : với một cây tổng quát cho, thì biểu diễn trong máy của nó là cây nhị phân tương đương. Các phép xử lí trên cây tổng quát đều thực hiện qua cây nhị phân tương đương này và kết quả sau khi chuyển đổi lại. sẽ phải khớp với ý đồ xử lí đối với cây tổng quát.

Sau đây là ví dụ minh họa một vài cây nhị phân tương đương ứng với cây tổng quát cho:

HÌNH 6.11



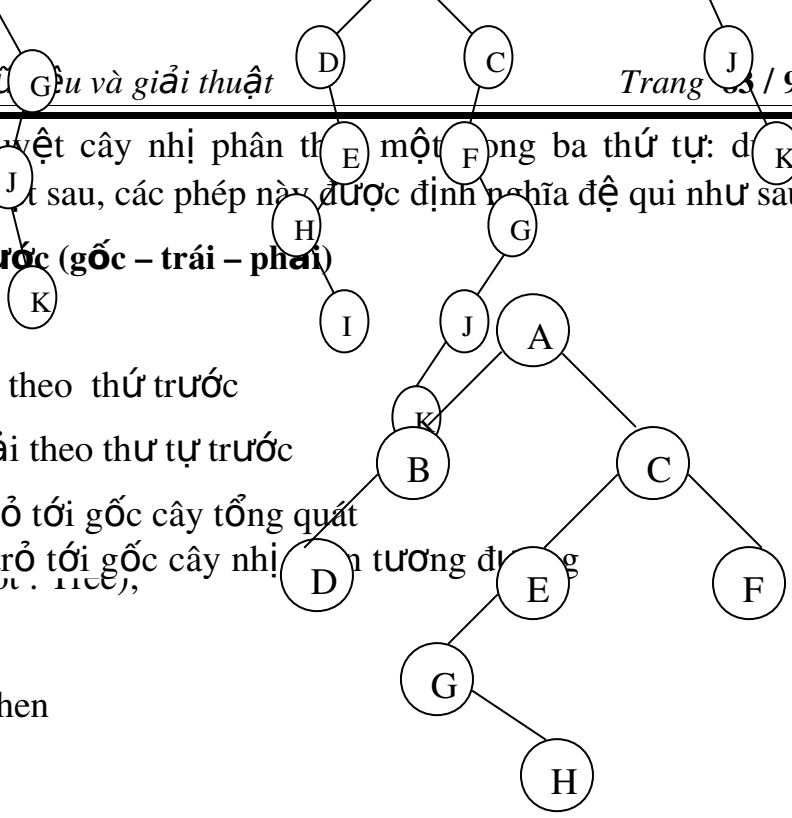
Ta thấy :

- Gốc của cây nhị phân tương đương T' không có con phải.
- Cây nhị phân tương đương T' của một cây nhị phân T khác nó.

3. Bài toán duyệt cây nhị phân

Phép xử lý các nút trên cây - mà ta gọi chung là phép thăm các nút một cách hệ thống, sao cho mỗi nút được thăm đúng một lần, gọi là phép duyệt

cây. Chúng ta thường duyệt cây nhị phân theo một trong ba thứ tự: duyệt trước, duyệt giữa và duyệt sau, các phép này được định nghĩa đệ qui như sau:



HÌNH 6.12

3.1. Duyệt theo thứ tự trước (gốc – trái – phải)

- Thăm gốc
- Duyệt cây con trái theo thứ trước
- Duyệt cây con phải theo thư tự trước

Cài đặt: T là con trỏ,Trỏ tới gốc cây tổng quát
T' la con trỏ, trỏ tới gốc cây nhị phân tương đương của cây T)

```
procedure Truoc(Root : Tree);
Begin
    if Root <> nil then
        Begin
            write(Root^.info);
            Truoc(Root^.left);
            Truoc(Root^.right);
        end;
 end;
```

Ví dụ: Chúng ta duyệt trước với cây ở hình 5.12, có kết quả như sau:

A B D C E G H F

3.2. Duyệt theo thứ tự giữa (trái – gốc – phải)

- Duyệt cây con trái theo thứ giữa
- Thăm gốc
- Duyệt cây con phải theo thư tự giữa

Cài đặt:

```
procedure Giua(Root : Tree);
Begin
    if Root^.left <> nil then
        Begin
            Preorder(Root^.left);
            write(Root^.info);
            Preorder(Root^.right);
```

```
            end;
    end;
```

Ví dụ: Chúng ta duyệt trước với cây ở hình 5.12, có kết quả như sau:

D B A G H E C F

3.3. Duyệt theo thứ tự sau (trái – phải – gốc)

- Duyệt cáycon trái theo thứ sau
- Duyệt cây con phải theo thư tự sau
- Thăm gốc

Cài đặt:

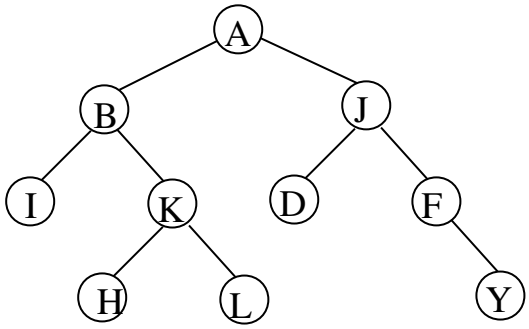
```
procedure Sau(Root : Tree);
Begin
    if Root^.right <> nil then
            Begin
                Preorder(Root^.left);
                Preorder(Root^.right);
                write(Root^.info);
            end;
    end;
```

Ví dụ: Chúng ta duyệt trước với cây ở hình 5.12, có kết quả như sau:

D B H G E F C A

Bài tập thực hành của học viên

6.1. Trình bày các biểu thức theo thứ tự duyệt trước, duyệt sau, duyệt giữa của cây sau:



6.2. Dựng cây nhị phân biết thứ tự các đỉnh khi duyệt theo

a. Thứ tự trước: A D F G H K L P Q R W Z

Thứ tự giữa : G F H K D L A W R Q P Z

b. Theo thứ tự sau: F G H D A L P Q R Z W K

Thứ tự giữa : G F H K D L A W R Q P Z

YÊU CẦU VỀ ĐÁNH GIÁ KẾT QUẢ HỌC TẬP:

Tiêu chí đánh giá	**Kết quả thực hiện**	**Hệ số**	**Kết qủa học tập**
Kiến thức		0,3	
Kỹ năng		0,5	
Thái độ		0,2	
Cộng:			

CHƯƠNG 7: ĐỒ THỊ

Mã chương: Mh17-07

Giới thiệu:

Đồ thị có vai trò rất quan trọng trong thực tế. Đồ thị được dùng để giải các bài toán trong nhiều lĩnh vực khác nhau. Ví dụ, dùng đồ thị để biểu diễn các mạch điện, biểu diễn các công thức phân tử hóa học, biểu diễn mạng máy tính. Đồ thị với các trọng số được gán cho các cạnh của nó có thể dùng để giải các bài toán như bài toán tìm đường đi ngắn nhất giữa hai thành phố trong một mạng giao thông. Chúng ta cũng có thể dùng đồ thị để lập lịch thi và phân chia kênh cho các đài truyền hình.

Mục tiêu:

- Trình bày được khái niệm về đồ thị;

- Cài đặt được đồ thị trên máy tính bằng các cấu trúc mảng và cấu trúc danh sách liên kết;

- Giải được bài toán tìm đường đi trên đồ thị.

- Thực hiện các thao tác an toàn với máy tính.

Nội dung gồm:

1.Các định nghĩa

Một đồ thị G bao gồm một tập hợp V các đỉnh và một tập hợp E các cung, ký hiệu G=(V,E). Các đỉnh còn được gọi là nút (node). Các cung nối giữa hai đỉnh, hai đỉnh này có thể trùng nhau.

Hai đỉnh có cung nối nhau gọi là hai đỉnh kề (adjacency).

Một cung nối giữa hai đỉnh v, w có thể coi như là một cặp điểm (v,w). Nếu cặp này có thứ tự thì ta có cung có thứ tự, ngược lại thì cung không có thứ tự.

Nếu các cung trong đồ thị G có thứ tự thì G gọi là đồ thị có hướng (directed graph).

Nếu các cung trong đồ thị G không có thứ tự thì đồ thị G là đồ thị vô hướng (undirected graph).

Trong các phần sau này ta dùng từ đồ thị (graph) để nói đến đồ thị nói chung, khi nào cần phân biệt rõ ta sẽ dùng đồ thị có hướng, đồ thị vô hướng.

Thông thường trong một đồ thị, các đỉnh biểu diễn cho các đối tượng còn các cung biểu diễn mối quan hệ (relationship) giữa các đối tượng đó. Chẳng

hạn các đỉnh có thể biểu diễn cho các thành phố còn các cung biểu diễn cho đường giao thông nối giữa hai thành phố.

Một đường đi (path) trên đồ thị là một dãy tuần tự các đỉnh v1, v2,..., vn sao cho (vi,vi+1) là một cung trên đồ thị (i=1,...,n-1). Đường đi này là đường đi từ v_1 đến v_n và đi qua các đỉnh v_2,...,v_{n-1}. Đỉnh v_1 còn gọi là đỉnh đầu, v_n gọi là đỉnh cuối. Độ dài của đường đi này bằng (n-1). Trường hợp đặc biệt dãy chỉ có một đỉnh v thì ta coi đó là đường đi từ v đến chính nó có độ dài bằng không.

Đường đi gọi là đơn (simple) nếu mọi đỉnh trên đường đi đều khác nhau, ngoại trừ đỉnh đầu và đỉnh cuối có thể trùng nhau. Một đường đi có đỉnh đầu và đỉnh cuối trùng nhau gọi là một chu trình (cycle). Một chu trình đơn là một đường đi đơn có đỉnh đầu và đỉnh cuối trùng nhau và có độ dài ít nhất là 1.

Trong nhiều ứng dụng ta thường gắn các giá trị (value) vào các cung thể hiện một thông tin liên quan tới cung đó, giá trị đó được gọi là trọng số, lúc này ta nói đồ thị có trọng số.

Đồ thị con của một đồ thị G=(V,E) là một đồ thị G'=(V',E') trong đó:

? V'⊆V và

? E' gồm tất cả các cạnh (v,w) ∈ E sao cho v,w ∈ V'.

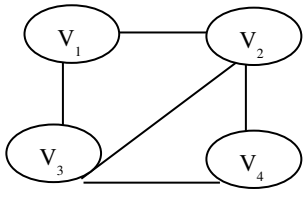
2. Biểu diễn đồ thị

2.1. Biểu diễn đồ thị bằng ma trận kề

Ta dùng một mảng hai chiều, chẳng hạn mảng A, kiểu boolean để biểu diễn các đỉnh kề. Nếu đồ thị có n đỉnh thì ta dùng mảng A có kích thước nxn. Giả sử các đỉnh được đánh số 1..n thì A[i,j] = true, nếu có cung nối giữa đỉnh thứ i và đỉnh thứ j, ngược lại thì A[i,j] = false. Rõ ràng, nếu G là đồ thị vô hướng thì ma trận kề sẽ là ma trận đối xứng.

Ví dụ 11: Ma trận liền kề với thứ tự các đỉnh v_1, v_2, v_3, v_4 là:

1	1	1	0
0	1	0	1
0	1	1	0
0	0	0	1



Do ma trận lưu trữ kế tiếp, nên việc truy cập đến các phần tử của ma trận kề được thực hiện trực tiếp, với tốc độ nhanh, nhưng cách lưu trữ này rỏ rang là tốn không gian nhớ khi n lớn

2.2. Biểu diễn đồ thị bằng danh sách các đỉnh kề:

Trong cách biểu diễn này, ta sẽ lưu trữ các đỉnh kề với một đỉnh i trong một danh sách liên kết theo một thứ tự nào đó, gọi là danh sách kề. Mỗi nút trong danh sách có qui cách:

INFO	LINK

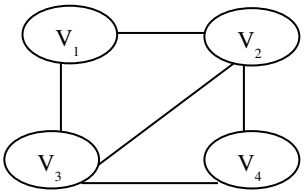
Trường INFO chứa số j ứng với đỉnh j là kề của i(1<=i<=n) trong danh sách lân cận của i

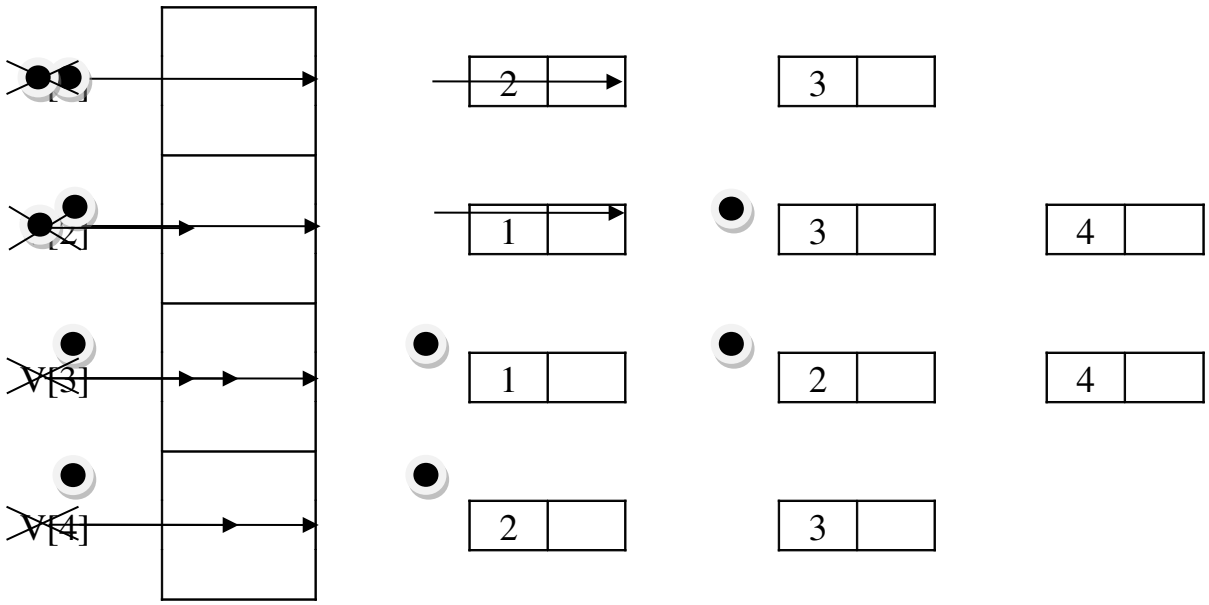
Trường LINK chứa con trỏ, trỏ tới nút tiếp theo trong danh sách.

Danh sách có m nút, nếu đỉnh i có m đỉnh kề .

Mỗi danh sách như vậy lại có "nút đầu danh sách". Thường các "nút đầu danh sách" là các phần tử của một vec tơ V, có kích thước bằng n, phần tử V[i] của V (1<=i<=n), ứng với danh sách kề của nút thứ i, nó chứa địa chỉ của nút đầu tiên của danh sách kề này.

Ví dụ 11: Hình ảnh danh sách kề biểu diễn đồ thị sau là:





3. Bài toán tìm đường đi trên đồ thị

Trong đời sống, chúng ta thường gặp những tình huống như sau: để đi từ địa điểm A đến địa điểm B trong thành phố, có nhiều đường đi, nhiều cách đi; có lúc ta chọn đường đi ngắn nhất (theo nghĩa cự ly), có lúc lại cần chọn đường đi nhanh nhất (theo nghĩa thời gian) và có lúc phải cân nhắc để chọn đường đi rẻ tiền nhất (theo nghĩa chi phí), v.v...

Có thể coi sơ đồ của đường đi từ A đến B trong thành phố là một đồ thị, với đỉnh là các giao lộ (A và B coi như giao lộ), cạnh là đoạn đường nối hai giao lộ. Trên mỗi cạnh của đồ thị này, ta gán một số dương, ứng với chiều dài của đoạn đường, thời gian đi đoạn đường hoặc cước phí vận chuyển trên đoạn đường đó, ...

Đồ thị có trọng số là đồ thị G=(V,E) mà mỗi cạnh (hoặc cung) e E được gán bởi một số thực m(e), gọi là trọng số của cạnh (hoặc cung) e.

Cho đơn đồ thị liên thông, có trọng số G=(V,E). Tìm khoảng cách $d(u_0,v)$ từ một đỉnh u_0 cho trước đến một đỉnh v bất kỳ của G và tìm đường đi ngắn nhất từ u_0 đến v.

Ở đây, chúng ta có thuật toán do E. Dijkstra, nhà toán học người Hà Lan, đề xuất năm 1959. Phương pháp của thuật toán Dijkstra là: xác định tuần tự đỉnh có khoảng cách đến u_0 từ nhỏ đến lớn.

Trước tiên, đỉnh có khoảng cách đến a nhỏ nhất chính là a, với $d(u_0,u_0)=0$. Trong các đỉnh v u_0, tìm đỉnh có khoảng cách k_1 đến u_0 là nhỏ nhất. Đỉnh này phải là một trong các đỉnh kề với u_0. Giả sử đó là u_1. Ta có: $d(u_0,u_1) = k_1$.

Trong các đỉnh v u_0 và v u_1, tìm đỉnh có khoảng cách k_2 đến u_0 là nhỏ nhất. Đỉnh này phải là một trong các đỉnh kề với u_0 hoặc với u_1. Giả sử đó là u_2. Ta có:

$$d(u_0,u_2) = k_2.$$

Tiếp tục như trên, cho đến bao giờ tìm được khoảng cách từ u_0 đến mọi đỉnh v của G. Nếu $V=\{u_0, u_1, ..., u_n\}$ thì:

$$0 = d(u_0,u_0) < d(u_0,u_1) < d(u_0,u_2) < ... < d(u_0,u_n).$$

Thuật toán *Dijkstra*

procedure *Dijkstra* (G=(V,E) là đơn đồ thị liên thông, có trọng số với trọng số dương)

{G có các đỉnh $a=u_0, u_1, ..., u_n=z$ và trọng số $m(u_i,u_j)$, với $m(u_i,u_j) =$ nếu (u_i,u_j) không là một cạnh trong G}

for i := 1 **to** n

$L(u_i)$:=

L(a) := 0

S := V \ {a}

u := a

while S

begin

for tất cả các đỉnh v thuộc S

if L(u) +m(u,v) < L(v) **then** L(v) := L(u)+m(u,v)

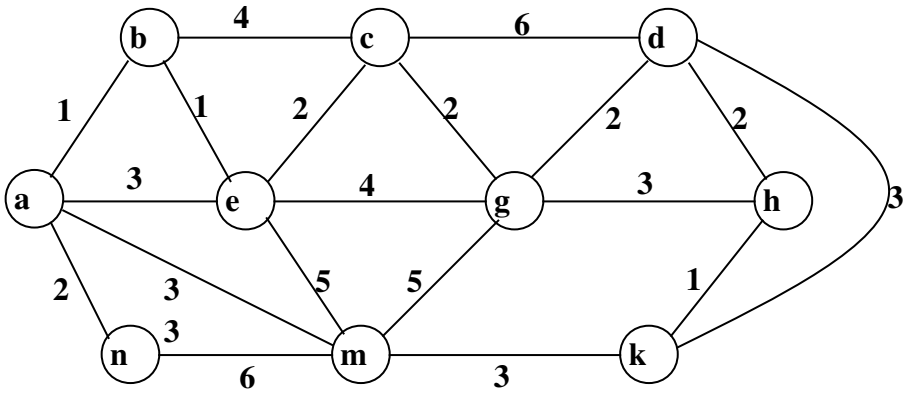
u := đỉnh thuộc S có nhãn L(u) nhỏ nhất

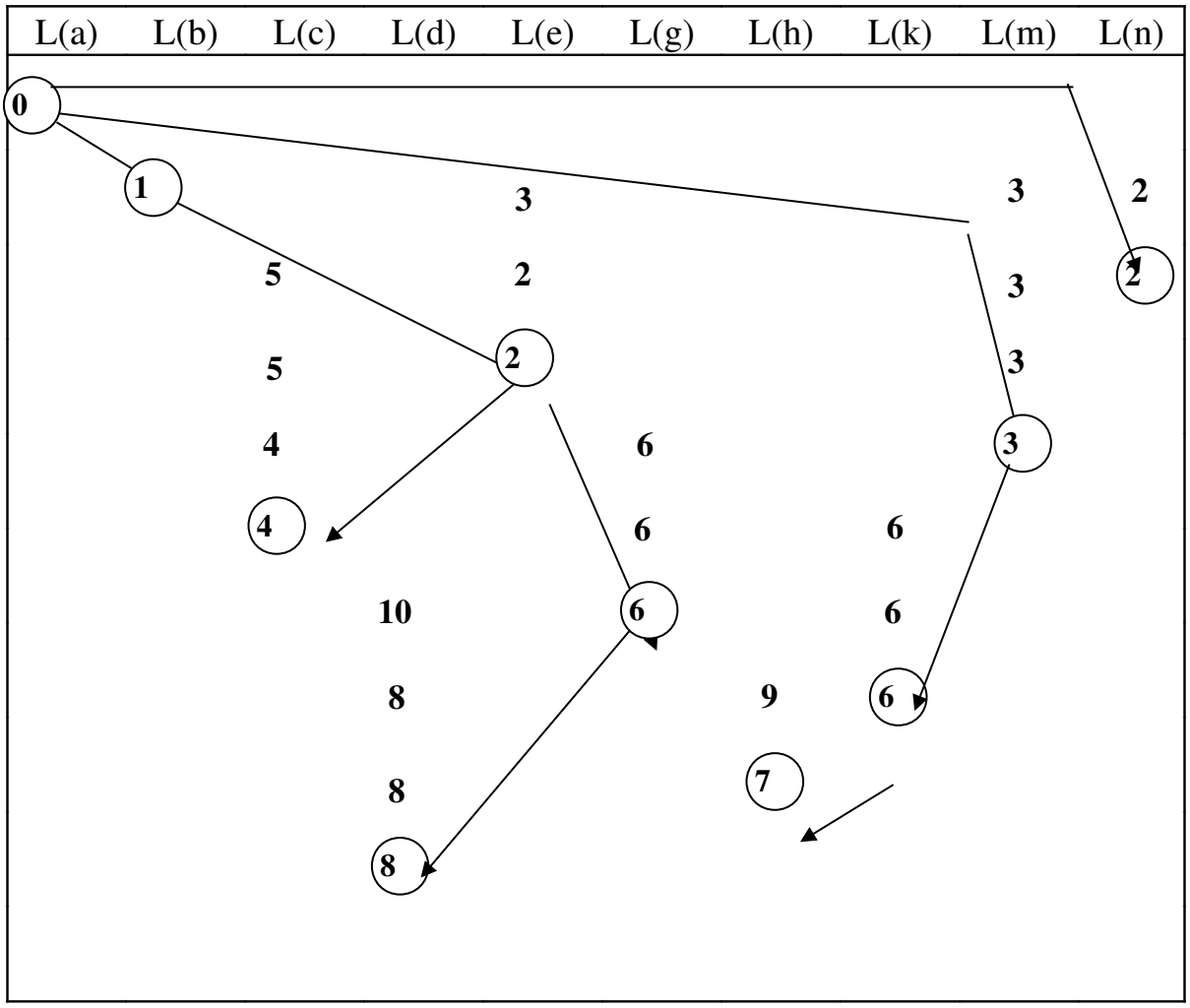
{L(u): độ dài đường đi ngắn nhất từ a đến u}

S := S \ {u}

End;

Ví dụ 1: Tìm khoảng cách d(a,v) từ a đến mọi đỉnh v và tìm đường đi ngắn nhất từ a đến v cho trong đồ thị G sau.





Bài tập thực hành của học viên

7.1. Hãy nêu vài ví dụ thực tế biểu diễn được bằng cấu trúc đồ thị.

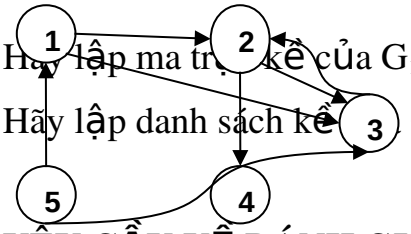
7.2. Cây có phải là một đồ thị ?

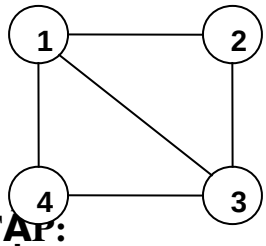
7.3. Một đồ thị vô hướng có 5 đỉnh thì có tối đa bao nhiêu cung giữa một đỉnh với một đồ thị đỉnh khác.

7.4. Với đồ thị có n đỉnh thì sao ?

7.5. Vẽ một đồ thị định hướng có 5 đỉnh A,B,C,D,E và các cung định hướng : (A,B);(C,D);(D,A);(B,D);(B,E);(E,D).

7.6. Cho các đồ thị G_1 và G_2 sau đây :





Hãy lập ma trận kề của G_1 và G_2

Hãy lập danh sách kề của G_1 và G_2

YÊU CẦU VỀ ĐÁNH GIÁ KẾT QUẢ HỌC TẬP:

Tiêu chí đánh giá	**Kết quả thực hiện**	**Hệ số**	**Kết qủa học tập**
Kiến thức		0,3	
Kỹ năng		0,5	
Thái độ		0,2	
Cộng:			

TÀI LIỆU THAM KHẢO

1. Aho, Hopcroft & Ullman, Data Structures and Algorithms,Addison Wesley, 2001.
2. Robert Sedewick, Algorithms in Java Third Edition,Addison Wesley, 2002.

   Niklaus Wirth, Data Structures and Algorithms,Prentice Hall, 2004.

   Robert Sedewick, Algorithms,Addison Wesley, 1983.
3. Bruno R. Preiss, Data Structures and Algorithms with Object-Oriented Design, Jon Wiley & Sons, 1998.
4. Đinh Mạnh Tường, Cấu trúc dữ liệu và thuật toán, NXB Đại học Quốc gia Hà Nội.
5. PGS.TS.Đỗ Xuân Lôi – Cấu trúc dữ liệu và giải thuật – NXB Khoa học và Kỹ thuật – 1997.
6. PGS.TS.Đỗ Xuân Lôi – Giáo trình Cấu trúc dữ liệu và giải thuật – Vụ giáo dục chuyên nghiệp, NXB Giáo dục - 2002
7. Trần Hạnh Nhi, *Giáo trình cấu trúc dữ liệu,* Trường đại học Khoa hoc tựnhiên, tp. Hồ Chí Minh, 2003
8. PGS. TS. Hoàng Nghĩa Tý, *Cấu Trúc Dữ Liệu Và Thuật Toán,* Xây Dựng, 2002
9. Gia Việt(Biên dịch), ESAKOV.J , WEISS T, *Bài Tập Nâng Cao Cấu Trúc Dữ Liệu Cài Đặt Bằng C*, Nhà xuất bản: Thống kê
10. Minh Trung (Biên dịch), TS. Khuất Hữu Thanh(Biên dịch), Chu Trọng Lương(Tác giả), *455 Bài Tập Cấu Trúc Dữ Liệu - Ứng Dụng Và Cài Đặt Bằng C++,* Thống kê .
11. Robert Sedgewick, Trần Đan Thư(Biên dịch), Bùi thị Ngọc Nga(Biên dịch), *Cẩm Nang Thuật Toán (Tập1,2)*; Khoa học và kỹ thuật
12. GS. TSKH. Hoàng Kiếm, *Giải Một Bài Toán Trên Máy Tính Như Thế Nào,* Giáo dục, 2005.
13. Nguyễn Quốc Lượng, Hoàng Đức Hải – Cấu trúc dữ liệu + giải thuật = chương trình – NXB Giáo dục – 1996.

DANH SÁCH BAN BIÊN SOẠN GIÁO TRÌNH DẠY NGHỀ
TRÌNH ĐỘ TRUNG CẤP, CAO ĐẲNG

Tên giáo trình: Cấu trúc dữ liệu và giải thuật

Tên nghề: Quản trị mạng

1. Bà Ngô Thị Thanh Trang	Chủ nhiệm
2. Ông Nguyễn Văn Hưng	Phó chủ nhiệm
3. Ông Trương Văn Hòa	Thư ký

DANH SÁCH HỘI ĐỒNG NGHIỆM THU
GIÁO TRÌNH DẠY NGHỀ TRÌNH ĐỘ TRUNG CẤP, CAO ĐẲNG

(font chữ Times New Roman, in hoa, cỡ chữ 14 Bold)

1. Ông (bà).....	Chủ tịch
2. Ông (bà).....	Phó chủ tịch
3. Ông (bà).....	Thư ký
4. Ông (bà).....	Thành viên
5. Ông(bà).....	Thành viên
6. Ông(bà).....	Thành viên
7. Ông(bà).....	Thành viên
8. Ông(bà).....	Thành viên
9. Ông(bà).....	Thành viên