SỞ GIÁO DỤC VÀ ĐÀO TẠO HÀ NỘI

GIÁO TRÌNH

Cấu trúc dữ liệu và giải thuật

DÙNG TRONG CÁC TRƯỜNG TRUNG HỌC CHUYÊN NGHIỆP

NHÀ XUẤT BẢN HÀ NỘI

SỞ GIÁO DỤC VÀ ĐÀO TẠO HÀ NỘI

NGUYỄN THÁI HÀ

GIÁO TRÌNH
CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

(Sách dùng trong các trường THCN Hà Nội)

NHÀ XUẤT BẢN HÀ NỘI - 2007

Lời giới thiệu

Nước ta đang bước vào thời kỳ công nghiệp hóa, hiện đại hóa nhằm đưa Việt Nam trở thành nước công nghiệp văn minh, hiện đại.

Trong sự nghiệp cách mạng to lớn đó, công tác đào tạo nhân lực luôn giữ vai trò quan trọng. Báo cáo Chính trị của Ban Chấp hành Trung ương Đảng Cộng sản Việt Nam tại Đại hội Đảng toàn quốc lần thứ IX đã chỉ rõ: "Phát triển giáo dục và đào tạo là một trong những động lực quan trọng thúc đẩy sự nghiệp *công nghiệp hóa, hiện đại hóa*, là điều kiện để phát triển nguồn lực con người - yếu tố cơ bản để phát triển xã hội, tăng trưởng kinh tế nhanh và bền vững".

Quán triệt chủ trương, Nghị quyết của Đảng và Nhà nước và nhận thức đúng đắn về tầm quan trọng của chương trình, giáo trình đối với việc nâng cao chất lượng đào tạo, theo đề nghị của Sở Giáo dục và Đào tạo Hà Nội, ngày 23/9/2003, Ủy ban nhân dân thành phố Hà Nội đã ra Quyết định số 5620/QĐ-UB cho phép Sở Giáo dục và Đào tạo thực hiện đề án biên soạn chương trình, giáo trình trong các trường Trung học chuyên nghiệp (THCN) Hà Nội. Quyết định này thể hiện sự quan tâm sâu sắc của Thành ủy, UBND thành phố trong việc nâng cao chất lượng đào tạo và phát triển nguồn nhân lực Thủ đô.

Trên cơ sở chương trình khung của Bộ Giáo dục và Đào tạo ban hành và những kinh nghiệm rút ra từ thực tế đào tạo, Sở Giáo dục và Đào tạo đã chỉ đạo các trường THCN tổ chức biên soạn chương trình, giáo trình một cách khoa học, hệ

thống và cập nhật những kiến thức thực tiễn phù hợp với đối tượng học sinh THCN Hà Nội.

Bộ giáo trình này là tài liệu giảng dạy và học tập trong các trường THCN ở Hà Nội, đồng thời là tài liệu tham khảo hữu ích cho các trường có đào tạo các ngành kỹ thuật - nghiệp vụ và đông đảo bạn đọc quan tâm đến vấn đề hướng nghiệp, dạy nghề.

Việc tổ chức biên soạn bộ chương trình, giáo trình này là một trong nhiều hoạt động thiết thực của ngành giáo dục và đào tạo Thủ đô để kỷ niệm "50 năm giải phóng Thủ đô", "50 năm thành lập ngành" và hướng tới kỷ niệm "1000 năm Thăng Long - Hà Nội".

Sở Giáo dục và Đào tạo Hà Nội chân thành cảm ơn Thành ủy, UBND, các sở, ban, ngành của Thành phố, Vụ Giáo dục chuyên nghiệp Bộ Giáo dục và Đào tạo, các nhà khoa học, các chuyên gia đầu ngành, các giảng viên, các nhà quản lý, các nhà doanh nghiệp đã tạo điều kiện giúp đỡ, đóng góp ý kiến, tham gia Hội đồng phản biện, Hội đồng thẩm định và Hội đồng nghiệm thu các chương trình, giáo trình.

Đây là lần đầu tiên Sở Giáo dục và Đào tạo Hà Nội tổ chức biên soạn chương trình, giáo trình. Dù đã hết sức cố gắng nhưng chắc chắn không tránh khỏi thiếu sót, bất cập. Chúng tôi mong nhận được những ý kiến đóng góp của bạn đọc để từng bước hoàn thiện bộ giáo trình trong các lần tái bản sau.

GIÁM ĐỐC SỞ GIÁO DỤC VÀ ĐÀO TẠO

Lời nói đầu

Tin học và viễn thông là hai thành phần cốt lõi của công nghệ thông tin. Trong những năm gần đây, nhiều dự án phát triển công nghệ thông tin ở nước ta đã được triển khai theo các giải pháp tổng thể trong đó tích hợp hạ tầng truyền thông máy tính với các chương trình tin học ứng dụng. Nhu cầu hiểu biết về tin học ngày càng cao và không chỉ dừng ở mức người sử dụng mà còn đi sâu hơn để làm chủ hệ thống và phát triển các phần mềm ứng dụng.

Cấu trúc dữ liệu và giải thuật *là một môn học cơ sở trong chương trình đào tạo kỹ thuật viên tin học. Để tạo điều kiện cho người học làm quen với một số kiến thức cơ bản về cấu trúc dữ liệu và giải thuật xử lý trên các cấu trúc dữ liệu đó, tạo điều kiện cho việc nâng cao thêm về kỹ thuật lập trình, về phân tích và phương pháp giải các bài toán.*

Giáo trình được viết thành 7 chương, bao gồm các vấn đề cơ bản phải giải quyết khi thiết kế và cài đặt các giải thuật.

Các chương 1, 2, 3 bổ sung thêm một số nhận thức về mối quan hệ giữa cấu trúc dữ liệu và giải thuật, các vấn đề về phân tích và thiết kế giải thuật và giải thuật đệ quy.

Chương 4, 5, 6 giới thiệu một số cấu trúc dữ liệu và giải thuật cơ bản, điển hình như mảng, danh sách, cây, đồ thị...

Chương 7 giới thiệu về sắp xếp và tìm kiếm, một yêu cầu xử lý rất phổ biến trong các bài toán và các ứng dụng tin học.

Mỗi chương, ngoài các kiến thức cơ bản, một số các giải thuật đã được viết thành các chương trình minh họa. Các chương trình được viết bằng ngôn ngữ Pascal hoặc C mô phỏng theo các giải thuật đã đưa ra và hoàn toàn đã được kiểm nghiệm thực hiện trên máy. Hy vọng rằng giáo trình sẽ giúp cho người học tiếp thu dễ dàng các kiến thức cần thiết, góp phần nâng cao trình độ của mình về tin học.

Xin chân thành cảm ơn ý kiến đóng góp của các đồng nghiệp, bạn bè trong quá trình biên soạn giáo trình này. Mặc dù tác giả đã rất cố gắng song chắc chắn không tránh khỏi những thiếu sót. Chúng tôi mong muốn nhận được các ý kiến đóng góp để hoàn thiện hơn nữa nội dung của giáo trình.

TÁC GIẢ

Bài mở đầu

1. Giải thuật và cấu trúc dữ liệu

Giải thuật là một dãy các câu lệnh được thao tác trên một số đối tượng. Sau một số hữu hạn các bước sẽ cho một kết quả mong muốn.

Giải thuật phản ánh phép xử lý còn đối tượng xử lý chính là các dữ liệu, chúng biểu diễn những thông tin cần thiết cho máy tính.

Không thể nói tới giải thuật mà không quan tâm tới vấn đề: giải thuật phải được tác động trên dữ liệu nào.

Còn dữ liệu khi xét tới cũng cần phải hiểu: dữ liệu đó sẽ được xử lý bằng giải thuật nào để được kết quả như mong muốn.

Các phần tử của dữ liệu có quan hệ với nhau nếu được "tổ chức" một cách thích hợp thì việc thực hiện các phép xử lý dữ liệu sẽ đơn giản, dễ dàng hơn.

Ta có thể nhận thấy rằng khi cấu trúc dữ liệu thay đổi dẫn đến giải thuật cũng cần thay đổi theo.

Ví dụ: Giả sử ta có có một danh sách các đối tượng: mỗi đối tượng gồm "Tên hàng, số lượng" (a_1,b_1) (a_2,b_2) (a_n,b_n). Viết chương trình sao cho khi biết "Tên hàng" máy tính sẽ cho ta biết "Số lượng" của mặt hàng đó. Đó là một lớp bài toán mà phép xử lý cơ bản là phép "Tìm kiếm". Ta hãy thử một vài phương pháp:

- Cách đơn giản là tìm từ đầu đến cuối danh sách nếu trùng tên hàng thì tìm được ra số lượng; như vậy khả năng xấu nhất là cần duyệt hết n đối tượng. Do vậy nếu n lớn thì rất mất thời gian.

- Nếu danh sách trên được sắp xếp (theo thứ tự từ điển chẳng hạn) thì việc tìm kiếm sẽ đơn giản hơn, vì ta sẽ áp dụng một giải thuật hiệu quả hơn trong trường hợp này.

- Nếu lại tổ chức thêm một bảng mục lục chỉ dẫn nữa như theo chữ cái đầu tiên của "Tên hàng" thì chắc chắn rằng khi cần tìm một mặt hàng chẳng hạn: "Ti vi", ta sẽ bỏ qua tất cả các mặt hàng mà tên không bắt đầu bằng ký tự 'T'; nói một cách cụ thể hơn ta sẽ tìm đến tất cả tên hàng có chữ T, sau đó trong các mặt hàng đó tìm tên hàng là "Ti vi".

Như vậy giữa cấu trúc dữ liệu và giải thuật có quan hệ mật thiết hai chiều. Chính điều này dẫn tới sự cần thiết phải nghiên cứu các cấu trúc dữ liệu đi đôi với việc xác lập các giải thuật tác động lên các cấu trúc đó.

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

Ta đề cập tới một số vấn đề như sau:

2.1. Dữ liệu bao gồm các phần tử cơ sở ta gọi là dữ liệu nguyên tử

(Dữ liệu nguyên tử có thể là một số, một từ, một bản ghi). Chúng có thể liên kết lại theo các cách khác nhau.

- Liên kết động
- Liên kết tĩnh.

Lựa chọn một cấu trúc dữ liệu thích hợp để tổ chức dữ liệu vào (input) và trên cơ sở đó xây dựng được giải thuật xử lý có hiệu quả là một bước quan trọng có tính quyết định.

Cần chú ý rằng trong những năm gần đây, khi mà các bài toán phi số xuất hiện ngày một nhiều, với những đặc tính phức tạp về phương diện dữ liệu, đòi hỏi chúng ta cần nghiên cứu sâu về cấu trúc dữ liệu.

2.2. Trong các bài toán phi số một loạt các phép xử lý mới xuất hiện như: phép tạo lập huỷ bỏ một cấu trúc, phép truy nhập vào từng phần tử của cấu trúc, phép bổ sung hoặc loại bỏ một phần tử trên cấu trúc. Các phép đó có tác dụng khác nhau đối với từng cấu trúc. Có phép hữu hiệu với cấu trúc này, không hữu hiệu với cấu trúc khác.

Vì vậy chọn cấu trúc dữ liệu phải nghĩ ngay các phép toán tác động cấu trúc dữ liệu ấy và ngược lại.

2.3. Cách biểu diễn một cấu trúc dữ liệu trong bộ nhớ được gọi là cấu trúc lưu trữ. Trên cơ sở lưu trữ này để thực hiện các phép xử lý phải phân biệt được cấu trúc dữ liệu và cấu trúc lưu trữ tương ứng. Có nhiều cách lưu trữ khác nhau với một cấu trúc dữ liệu, cũng như có thể có nhiều kiểu cấu trúc dữ liệu khác nhau được lưu trữ trong bộ nhớ cùng một kiểu cấu trúc lưu trữ.

Khi đề cập đến cấu trúc lưu trữ ta cần phân biệt cấu trúc lưu trữ tương ứng với bộ nhớ trong - lưu trữ trong và cấu trúc lưu trữ ứng với bộ nhớ ngoài - lưu trữ ngoài; chúng có những đặc điểm riêng nên sẽ kéo theo các cách xử lý khác nhau.

2.4. Trong một ngôn ngữ lập trình bao giờ cũng có một cấu trúc dữ liệu tiền định. Ví dụ cấu trúc mảng để tổ chức các tập dữ liệu có số lượng nhất định và cùng kiểu. Nếu cấu trúc dữ liệu tiền định phù hợp với cấu trúc dữ liệu của người

dùng thì tốt nhất. Nhưng nếu cấu trúc dữ liệu tiền định không được sử dụng thì rất khó khăn:

Chẳng hạn để lưu trữ hồ sơ cán bộ ta dùng các bản ghi. Trong một bản ghi có nhiều trường, mỗi trường là một kiểu dữ liệu. Với PASCAL ta dễ dàng thực hiện được. Còn với FORTRAN thì gặp khó khăn vì nó mô phỏng các mục dữ liệu dưới dạng véctơ hay ma trận.

Như vậy chấp nhận một ngôn ngữ lập trình tức là chấp nhận cấu trúc dữ liệu kiểu tiền định.

Ba vấn đề đầu tiên chính là mục tiêu đề cập đến của chúng ta trong giáo trình này.

3. Ngôn ngữ diễn đạt giải thuật

Ngôn ngữ được chọn phải luôn luôn tuân thủ khả năng xử lý của ngôn ngữ với bài toán đặt ra sao cho đơn giản nhất phù hợp với cấu trúc dữ liệu tiền định.

Trong môn học này dùng ngôn ngữ tựa PASCAL để mô phỏng *nhắc lại kiến thức cơ bản PASCAL.*

3.1. Quy cách về cấu trúc chương trình

Mỗi chương trình đều bắt đầu bằng từ khoá **program** và được gán một tên để phân biệt, tên có độ dài không hạn chế; ví dụ:

program tong_2_so;

Tất cả các chú giải được đặt trong cặp {...}

Chương trình bao gồm nhiều bước, mỗi bước sẽ được đánh số.

3.2. Ký tự và biểu thức

Giống như ngôn ngữ PASCAL.

3.3. Các câu lệnh

Các câu lệnh trong chương trình được viết cách nhau bởi dấu chấm phẩy (;) chúng bao gồm:

- Lệnh gán có dạng b: = bth;

Với *b* chỉ tên biến, tên hàm, còn *bth* chỉ biểu thức

Ở đây cho phép viết phép gán kép dạng x: = y: = c;

- Câu lệnh ghép

Có dạng: **begin** s1;s2;...;sn; **end**;

- Câu lệnh điều kiện

Có dạng: **if** B **then** S hoặc **if** B **then** S **else** S2;

Với B là biểu thức điều kiện còn S, S1, S2 là một lệnh khác

- Câu lệnh tuyển

Có dạng:

Case

B1: S1;

B2: S2:

.

.

Bn: Sn;

else: $\mathbf{S_{n+1}}$

end case

Ở đây B_i là các điều kiện, S_i là các câu lệnh

- Các lệnh lặp (giống như PASCAL)

- Câu lệnh nhảy: Có dạng goto n (n là nhãn của một bước trong chương trình;

- Câu lệnh vào, ra: Có dạng giống như PASCAL.

- Lệnh kết thúc: **end**

3.4. Chương trình con

3.4.1. Chương trình con hàm

Có dạng:

function tên_hàm (danh sách tham số)

s1; s2;...; sn;

return

Câu lệnh kết thúc chương trình con là **return** thay cho **end**

3.4.2. Chương trình con thủ tục

Giống như chương trình con hàm nhưng khác ở chỗ:

- Từ khoá **procedure** thay cho **function**

- Hàm bao giờ cũng được trả lại kết quả, thể hiện ở lệnh gán mà tên hàm nằm ở vế trái, giá trị nằm ở vế phải, còn thủ tục thì không.

Lời gọi chương trình con hàm được thể hiện thông qua tên hàm cùng danh sách tham số thực. Còn với chương trình con thủ tục lời gọi có dạng:

call tên thủ tục (danh sách tham số thực);

Chú ý: Trong các chương trình diễn đạt một giải thuật phần khai báo dữ liệu sẽ được bỏ qua; nó được thay thế bởi phần mô tả dữ liệu bằng ngôn ngữ tự nhiên.

Chương 1

THIẾT KẾ VÀ PHÂN TÍCH GIẢI THUẬT

Mục tiêu:

- Giúp học sinh hiểu được về giải thuật và ý nghĩa của giải thuật để giải một bài toán bằng chương trình.

- Người học biết được quy trình xây dựng lời giải một bài toán. Đó là chia một bài toán thành nhiều bài toán nhỏ để giải rồi sau đó phối hợp, ghép các bài toán nhỏ lại với nhau đưa trở về bài toán ban đầu.

- Người học biết các phương pháp thiết kế giải thuật để giải các bài toán liên quan.

- Người học biết được một số khái niệm liên quan đến giải thuật như thời gian thực hiện giải thuật T(n); độ phức tạp tính toán của giải thuật; và các yếu tố liên quan đến thời gian thực hiện giải thuật như kích thước của dữ liệu vào và ra của giải thuật; kiểu dữ liệu; tốc độ xử lý của máy tính; chương trình dịch; ngôn ngữ lập trình;...

- Nắm được các quy tắc Tổng; quy tắc Nhân để xác định độ phức tạp tính toán của giải thuật; ngoài ra chương này còn cung cấp một số ví dụ mẫu để người học củng cố kiến thức.

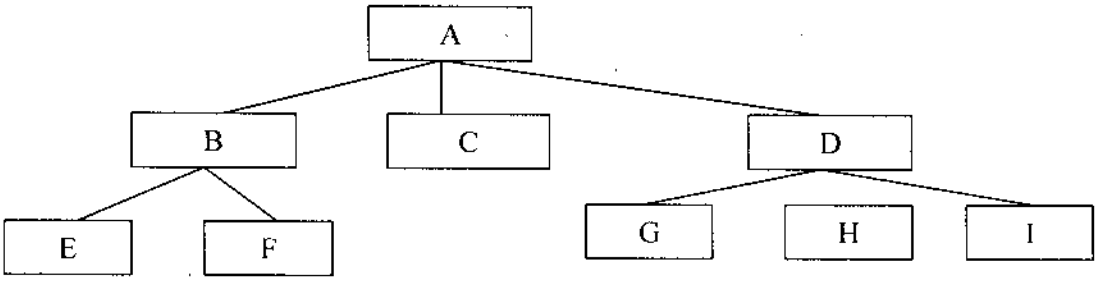
I. TỪ BÀI TOÁN ĐẾN CHƯƠNG TRÌNH

1. Modul hoá việc giải quyết bài toán

Các bài toán giải được trên máy tính điện tử ngày càng đa dạng và phức tạp; lẽ dĩ nhiên là các giải thuật và chương trình để giải chúng cũng ngày càng có quy mô và càng khó khi xây dựng cũng như khi tìm hiểu chúng.

Tuy nhiên mọi việc sẽ trở nên dễ dàng hơn nếu như ta phân chia bài toán lớn thành các bài toán nhỏ. Điều đó có nghĩa là nếu ta coi bài toán ban đầu như là một modul chính thì ta cần chia nó thành các modul con và đến lượt chúng các modul con này lại được phân chia tiếp thành các modul bé hơn,... cứ như thế ta sẽ thu được các modul ứng với các bài toán cơ bản mà ta biết cách giải

quyết. Như vậy việc giải quyết bài toán sẽ được thể hiện theo một cấu trúc phân cấp có dạng như mô hình sau:



Chia một bài toán phức tạp thành các phần đơn giản theo tinh thần "Chia để trị" > Để thể hiện tính chiến thuật đó người ta dùng chiến lược thiết kế từ đỉnh xuống (Top - Down); thiết kế từ khái quát đến chi tiết.

Để minh hoạ ta xét bài toán sau: *Xây dựng một chương trình quản lý học bổng cho một trường đại học.*

Trước hết ta phải hình dung được cái vào cái ra của bài toán:

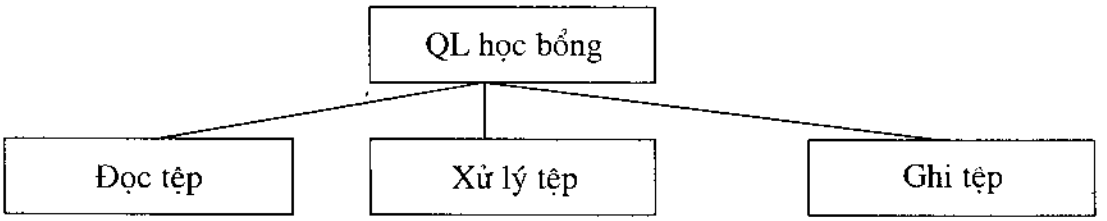
Có thể giả sử rằng ta đã có trong tay một tập hồ sơ (mà ta gọi là tệp) bao gồm các bản ghi về các thông tin liên quan tới học bổng của sinh viên, chẳng hạn: số hiệu sinh viên, điểm trung bình,...; chương trình phải thực hiện được các công việc sau:

1. Tìm và hiển thị được bản ghi của bất kỳ sinh viên nào.
2. Cập nhật bản ghi của mọi sinh viên nếu cần.
3. In các loại báo cáo, thống kê.

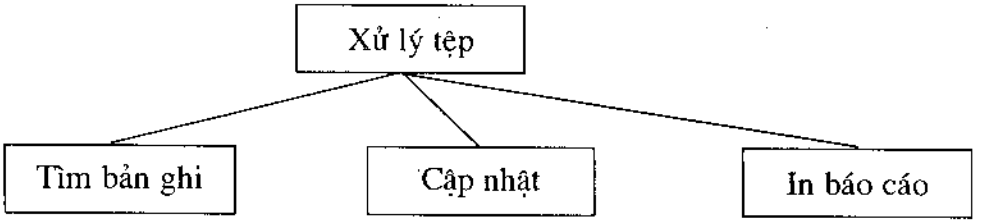
Như vậy giải thuật cần đáp ứng được 3 yêu cầu sau:

1. Phải đọc được các thông tin về mọi sinh viên đã được lưu trữ trên bộ nhớ ngoài (gọi là "Đọc tệp").
2. Xử lý các thông tin này (gọi là "Xử lý tệp").
3. Cập nhập các thông tin khi có sự thay đổi (gọi là "Ghi tệp").

Ta có thể biểu diễn ý đồ này như sau:



Các nhiệm vụ này lại được phân nhỏ tiếp; chẳng hạn, nhiệm vụ "Xử lý tệp" sẽ được phân thành ba yêu cầu như sau:



...

Cách thiết kế giải thuật theo kiểu Top - Down như thế giúp cho việc giải quyết bài toán một cách thuận lợi có định hướng, không bị sa đà vào các chi tiết phụ; nó cũng chính là nền tảng cho *lập trình cấu trúc*.

Mặt khác đối với một bài toán lớn, để giải được nó phải có nhiều người tham gia; chính phương pháp modul hoá sẽ cho phép tách bài toán ra nhiều phần độc lập tạo điều kiện cho nhiều nhóm chuyên gia chủ động giải quyết phần việc của mình mà không làm ảnh hưởng tới các nhóm khác.

2. Phương pháp tinh chỉnh từng bước

Là phương pháp thiết kế giải thuật kèm với lập trình. Tức là modul hoá và thiết kế kiểu Top - Down.

Thoạt đầu chương trình được thể hiện bằng ngôn ngữ tự nhiên sau đó được chi tiết hoá dần dần ta gọi là các bước tinh chỉnh. Càng về sau các lời lẽ được thay thế dần bằng các câu lệnh của ngôn ngữ lập trình. Ở giai đoạn trung gian dùng cả ngôn ngữ tự nhiên. Với ngôn ngữ lập trình gọi là giả ngôn ngữ. Như vậy quá trình thiết kế giải thuật bắt đầu từ:

- Ngôn ngữ tự nhiên.
- Giả ngôn ngữ.
- Ngôn ngữ lập trình.

Từ mức "Làm cái gì" → "Làm như thế nào"

Ví dụ 1: Sắp xếp một dãy số gồm n số nguyên theo thứ tự tăng dần. Có thể phác hoạ giải thuật như sau:

Từ dãy số nguyên *chưa được* sắp xếp *lấy ra* số nhỏ nhất rồi *đặt* nó vào cuối dãy *đã được* sắp xếp. Cứ làm như thế cho đến khi dãy chưa sắp xếp trở thành dãy rỗng.

Đặt vấn đề: Dãy số đã sắp xếp đặt ở chỗ cũ hay chỗ mới?

Ta quy ước dãy đã sắp xếp vẫn để ở chỗ cũ; điều này có nghĩa là việc đặt "số nhỏ nhất" vừa lấy ra (ở một lượt nào đó) vào cuối dãy đã được sắp xếp phải được thực hiện bằng cách đổi chỗ với số đang ở vị trí đó.

Giải thuật như sau:

For i:=1 to n do begin

- Xét từ a_i đến a_n; tìm số nhỏ nhất a_j
- Đổi chỗ a_i và a_j

End;

Tới đây có hai nhiệm vụ con:

1. Tìm số nguyên nhỏ nhất a_j trong các số từ a_i tới a_n
2. Đổi chỗ a_i và a_j

Nhiệm vụ đầu có thể thực hiện bằng cách:

Gọi a_i là số nhỏ nhất sau đó so sánh với a_{i+1}, a_{i+2},... Nếu thấy số nào nhỏ hơn thì coi nó là số nhỏ nhất. Khi đã so sánh với a_n rồi thì số nhỏ nhất được xác định. Để xác định được số nhỏ nhất ta chỉ cần nắm được chỉ số của phần tử ấy.

Ta có bước tinh chỉnh 1:

```
j:=i
For k:=j+1 to n do
    If a[k] < a[j] then j:=k;
```

Nhiệm vụ thứ 2 là đổi chỗ:

Ta có bước tinh chỉnh 2 (đổi chỗ):

```
B:=a[i]; a[i]:= a[j]; a[j]:=B;
```

Chương trình sắp xếp dưới dạng thủ tục sau:

```
Procedure SORT(A,n);
1. For i:=1 to n do begin
2.{Chọn số nhỏ nhất} j:=i
    For k:=j+1 to n do
        if A[k]<A[j] then j:=k;
3.{Đổi chỗ} B:=A[i]; A[i]:=A[j]; A[j]:=B; end;
4. Return.
```

Sau đây là chương trình hoàn chỉnh:

```
PROGRAM SAP_XEP;
uses crt;
const
   max=100;
type
   mang=array[1..max] of integer;
var
   day_so:mang;
   n, i, j, k: integer;
   tg: integer;
BEGIN
   clrscr;
   write ('Nhap so phan tu day so n= ');readln (n);
   writeln;
   writeln ('Nhap cac phan tu cua day:');
   for i:=1 to n do
     begin
         write('phan tu thu ',i,'= ');
         readln(day_so[i]);
     end;
   clrscr;
   writeln('Day so truoc khi sap xep:'); writeln;
   for i:=1 to n do
          write (day_so[i]:4);writeln;
   {doan chuong trinh sap xep}
   for i:=1 to n-1 do
      Begin
         j:=i;
         for k:=j+1 to n do
             if day_so [k] < day_so [j] then j:=k;
         if j<>i then Begin
                        tg:=day_so[i];
                        day_so[i]:=day_so[j];
                        day_so[j]:=tg;
```

```
                End;
        End;
    writeln ('Day so sau khi sap xep:'); writeln;
    for i:=1 to n do
          write (day_so[i]:4); writeln;
    readln;
END.
```

Ví dụ 2: Cho một ma trận vuông n = 3

3 2 5
6 7 1
8 2 0

In ra các phần tử thuộc các đường chéo // đường chéo chính.

Giả sử in từ trái qua phải, kết quả phải là:

5
2 1
3 7 0
6 2
8

Ta hướng vào chương trình PASCAL

Giải thuật phác hoạ như sau:

1. Nhập n
2. Nhập các phần tử của ma trận
3. In các đường chéo // đường chéo chính.

Nhiệm vụ (1) và (2):

1. Readln(n);
2. For i:=1 to n do
   For j:=1 to n do Readln (a[i,j]);

Nhiệm vụ 3: Phân làm hai loại

- Đường chéo ứng với cột từ n đến 1
- Đường chéo ứng với hàng từ 2 đến n

Vậy ta tách làm 2 nhiệm vụ con:

3.1 For j:=n downto 1 do
in đường chéo ứng với cột j

3.2 For i:=1 to n do

in đường chéo ứng với hàng i

Tới đây phải chi tiết

"In đường chéo ứng với cột j"

j=n in 1 phần tử hàng 1 cột j

j=n-1 in 2 phần tử hàng 1 cột j

hàng 2 cột j+1

j=n-2 in 3 phần tử hàng 1 cột j

hàng 2 cột j+1

hàng 3 cột j+2.

Số lượng các phần tử được in ra chính là (n-j+1); còn phần tử được in chính là A[i,j+(i-1)] với i lấy giá trị từ 1 tới (n-j+1).

Vậy 3.1 có thể tinh chỉnh:

```
For i:=1 to (n-j+1) do
        Write(a[i,j+(i-1)]);
        Writeln;
3.2 For j:=1 to n-i+1 do
        Write(a[i+j-1,j]);
        Writeln;
```

Sau đây là chương trình hoàn chỉnh:

```
PROGRAM IN_DUONG_CHEO_MA_TRAN;
Const max=10;
Type  mt=array [1..max,1..max] of integer;
Var
     a:mt;
     i, j, n :integer;
BEGIN
Repeat
     Write ('Nhập kích thước n ma trận A');
     Readln(n);
Until (0<n) and (n<=max);
Writeln ('Nhập các phần tử:');
For i:=1 to n do
```

```
        For j:=1 to n do
            Readln (a[i,j]);
            Writeln ('In đường chéo');
        For j:=n downto 1 do begin
            for i:=1 to n-j+1 do
            Writeln (a[i,j+i-1]);
            writeln;
        end;
    For i:=2 to n do Begin
        For j:=1 to n-i+1 do Writeln (a[i+j-1,j]);
        Writeln;
    end;
    END.
```

II. PHÂN TÍCH GIẢI THUẬT

Yêu cầu: - Tính đúng đắn của giải thuật
- Tính đơn giản.

Phân tích thời gian thực hiện giải thuật.

Một bài toán có nhiều cách giải xong chọn giải thuật nào mà thời gian được thực hiện nhanh nhất.

Thời gian thực hiện giải thuật phụ thuộc vào rất nhiều yếu tố:

- Kích thước dữ liệu đưa vào: Nếu số lượng dữ liệu đưa vào là n thì thời gian thực hiện giải thuật như một hàm của n: T(n).

- Các kiểu lệnh, tốc độ xử lý của máy tính, ngôn ngữ viết chương trình, chương trình dịch ảnh hưởng đến T(n). Như vậy không thể biểu diễn T(n) bằng phút, giây,... nhưng cũng không phải là không thể so sánh được các giải thuật về mặt tốc độ.

Nếu thời gian thực hiện một giải thuật này là $T_1(n) = C_n^2$ còn một giải thuật khác là $T_2 = Kn$, ở đây C, K là hằng số; khi đó nếu n khá lớn thì $T_2(n) < T_1(n)$. (Nếu n nhỏ thì không có ý nghĩa).

Cách đánh giá thời gian thực hiện giải thuật độc lập với máy tính và các yếu tố liên quan dẫn tới khái niệm về "cấp độ lớn của thời gian thực hiện giải thuật" hay còn gọi là "độ phức tạp tính toán của giải thuật".

III. ĐỘ PHỨC TẠP TÍNH TOÁN CỦA GIẢI THUẬT

Hiệu suất thời gian của một giải thuật thường được xem là một yếu tố quan trọng nhất; với hiệu suất thời gian được đo như thế nào; thời gian thực hiện một giải thuật phụ thuộc vào nhiều yếu tố. Đương nhiên một yếu tố quan trọng là kích thước đầu vào, bởi vì số lượng mục vào sẽ ảnh hưởng tới thời gian xử lý chúng; chẳng hạn thời gian cần thiết để sắp thứ tự một danh sách các mục chắc chắn phụ thuộc vào số lượng các mục trong danh sách. Như vậy thời gian thực hiện (T) một giải thuật phải được biểu diễn như hàm T(n) của độ lớn đầu vào n.

Các kiểu lệnh và tốc độ của máy tính cũng ảnh hưởng tới thời gian thực hiện; tuy nhiên những yếu tố này phụ thuộc vào máy tính đang sử dụng, vì vậy chúng ta không thể biểu diễn giá trị của T(n) một cách đầy đủ bằng các đơn vị thời gian chẳng hạn như giây. *Thay vào đó T(n) sẽ được tính gần đúng như là số các lệnh thực hiện.*

Một yếu tố khác có ảnh hưởng tới thời gian tính toán là chất lượng của chương trình do bộ dịch tạo ra. Không phải tất cả các bộ dịch đều tạo ra những chương trình có hiệu suất như nhau, từ đó suy ra rằng T(n) không thể được tính như là số các lệnh của máy tính, mà T(n) sẽ được tính như số lần thực hiện các lệnh trong giải thuật.

Để minh hoạ ta xét giải thuật sau: tìm giá trị trung bình của n số.

1. Đọc *n*
2. Khởi động *Sum* tại 0.
3. Khởi động *i tại* 0.
4. while $i < n$ thực hiện các bước sau:
   a. Đọc số
   b. Thêm số vào *Sum*.
   c. Tăng thêm 1 vào i
5. Tính *tb=Sum/n*

Ta thấy

Lệnh	Số lần thực hiện
1	1
2	1
3	1
4	n+1

5	n
6	n
7	n
8	1
. Tổng cộng	4n+5

Ta thấy thời gian thực hiện giải thuật này là:

$$T(n) = 4n+5$$

Khi giá trị của n tăng, T(n) cũng tăng một cách tuyến tính; chúng ta nói rằng T(n) có "bậc n", điều này được ký hiệu theo "ký pháp chữ O lớn":

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

Một cách tổng quát, thời gian tính toán T(n) của giải thuật được gọi là "có bậc f(n)" ký hiệu bởi:

$$T(n) = O(f(n))$$

Nếu tồn tại các số dương C và n_0 sao cho:

$$T(n) <= Cf(n) \text{ với mọi } n >= n_0$$

Độ phức tạp của giải thuật này được gọi là O(f(n)); ví dụ độ phức tạp của giải thuật mô tả ở trên là O(n) vì thời gian tính được là:

$$T(n) = 4n+5$$

Và vì:

$$4n+5 <= 4n+n \text{ với mọi } n >= 5$$

ta có: $T(n) <= 5n$ với mọi $n >= 5$

Như vậy ta chọn f(n) = n, $n_0 = 5$ và C = 5, ta có thể viết T(n) = O(n)

Nếu thời gian thực hiện giải thuật là $T(n) = Cn^2$ (C là hằng số) ta nói độ phức tạp tính toán của giải thuật này là cấp n^2.

$$T(n) = O(n^2)$$

Tổng quát ta có thể định nghĩa: một hàm f(n) được xác định là O(g(n))

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

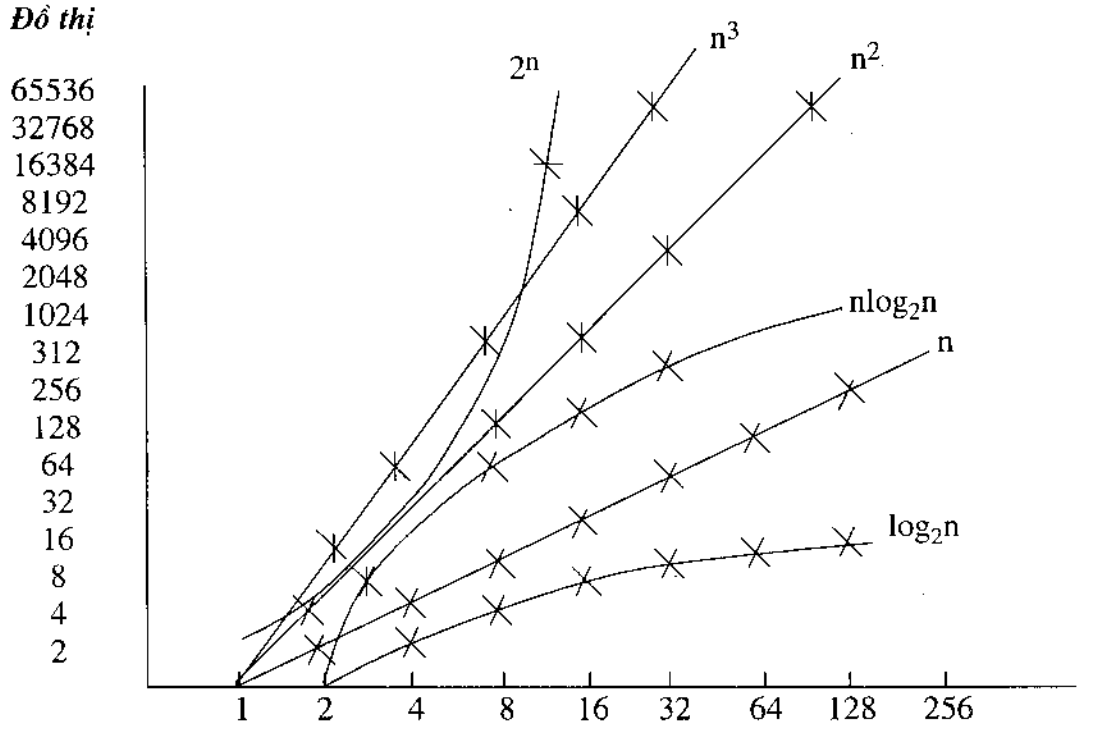
Và được gọi là cấp g(n) nếu tồn tại các hằng số c và n_0 sao cho:

$$f(n) <= cg(n) \text{ khi } n >= n_0$$

Thông thường các hàm thể hiện mức độ phức tạp tính toán của giải thuật có dạng: $\log_2 n$, n, $n\log_2 n$, n^2, n^3, 2^n, n!, n^n

Các hàm 2^n, n!, n^n được gọi là hàm số mũ, số còn lại gọi là các hàm loại đa thức.

Đồ thị



Log_2n	n	$nLog_2n$	n^2	n^3	2^n
0	1	0	1	1	2
1	2	2	4	8	4
2	4	8	16	64	16
3	8	24	64	512	256
4	16	96	256	4096	65536
5	32	160	1024	2768	2147483648

IV. XÁC ĐỊNH ĐỘ PHỨC TẠP TÍNH TOÁN

Xác định độ phức tạp tính toán của một giải thuật bất kỳ có thể dẫn tới những bài toán phức tạp. Tuy nhiên trong thực tế, đối với một số giải thuật ta cũng có thể phân tích được bằng một số quy tắc đơn giản.

1. Quy tắc tổng

Giả sử $T_1(n)$ và $T_2(n)$ là thời gian thực hiện của hai chương trình P1 và P2 mà $T_1(n) = O(f(n))$; $T_2(n) = O(g(n))$ thì thời gian thực hiện P1 rồi P2 tiếp theo

sẽ là: $T_1(n)+T_2(n) = O(\max(f(n),g(n))$.

Ví dụ: Một chương trình có 3 bước thực hiện: Thời gian thực hiện $O(n^2)$; $O(n^3)$ và $O(n\log^2 n)$ thì thời gian thực hiện hai bước đầu sẽ là $O(\max(n^2,n^3)) = O(n^3)$. Thời gian thực hiện chương trình sẽ là: $O(n^3,n\log_2 n) = O(n^3)$.

Một ứng dụng khác của quy tắc này là nếu $g(n) <= f(n)$; với $\forall n >= n_0$ thì $O(f(n)+g(n)) = O(j(n))$; chẳng hạn:

$$O(n^4 + n^2) = O(n^4),\ O(n+\log_2 n) = O(n).$$

2. Quy tắc nhân

Nếu tương ứng với P_1 và P_2 là $T_1(n) = O(f(n))$; $T_2(n) = O(g(n))$, thì thời gian thực hiện P_1 và P_2 lồng nhau sẽ là:

$$T_1(n).T_2(n) = O(f(n).g(n))$$

Ví dụ: câu lệnh gán x: = x+1 có thời gian thực hiện bằng hằng số C nên định giá là O(1), suy ra:

* Câu lệnh For i: = 1 to n do x: = x+1

Có thời gian thực hiện O(n.1) = O(n).

* Câu lệnh For i: = 1 to n do

For j: = 1 to n do x: = x+1

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

Chú ý: Khi đánh giá thời gian thực hiện giải thuật chỉ cần chú ý tới các bước tương ứng với một phép toán gọi là **phép toán tích cực**. Đó là phép toán thuộc giải thuật mà thời gian thực hiện nó không ít hơn thời gian thực hiện các phép khác, hay nói một cách khác số lần thực hiện nó không kém hơn các phép khác.

Ví dụ: Tính

$e^x \approx 1 + x/1! + x^2/2! + ... + x^n/n!$ Với x và n cho trước.

```
Program C1;
{Tính từng số hạng rồi cộng lại}
1. Read(x); S:=1;
2. For i:=1 to n do
   Begin
     P:=1;
     For j:=1 to i do P:=P*x/j
           S:=S+P;
   End;
```

3. **Return**.

Ta có thể coi phép toán tích cực ở đây là: P: = P*x/j; ta thấy nó thực hiện 1+2+3+...+n = n(n+1)/2 lần. Vậy thời gian thực hiện giải thuật là $T(n) = O(n^2)$

Ta có thể dùng giả thuật khác để giải quyết bài toán này:

Program C2;

{Dựa vào số hạng trước tính số hạng sau theo cách $x^n/n! = x^{n-1}/(n-1)! * x/n$*}*

1. **Read**(x); S:=1; P:=1;
2. **For** i:=1 **to** n **do begin**

   P:=P*x/i;

   S:=S+P;

   end;
3. **Return**

Bây giờ thời gian thực hiện giải thuật là T(n) = O(n) vì phép P: = P*x/i chỉ thể hiện n lần.

Chú ý: Thời gian thực hiện giải thuật không phải chỉ phụ thuộc vào kích thước dữ liệu mà còn phụ thuộc vào tình trạng dữ liệu đó.

Chẳng hạn: Sắp xếp một dãy số theo thứ tự tăng dần nếu gặp dãy số đưa vào có đúng thứ tự sắp xếp rồi khác với dãy số đưa vào chưa thứ tự sắp xếp hoặc có thứ tự ngược lại. Lúc đó khi phân tích thời gian thực hiện giải thuật ta phải xét tới: đối với dữ liệu có kích thước n thì T(n) trong trường hợp thuận lợi nhất, T(n) trong trường hợp xấu nhất, T(n) trong trường hợp trung bình.

Việc xác định T(n) trung bình dĩ nhiên là khó, phải dùng đến công cụ toán học đặc biệt hơn nữa. Tính trung bình có thể có nhiều cách quan niệm. Trong trường hợp T(n) trung bình khó xác định người ta thường đánh giá qua giải thuật xấu nhất của T(n).

Ví dụ: Cho vectơ V có n phần tử, tìm trong V giá trị bằng X cho trước.

1. Tim:=false; *{Báo hiệu ngừng tìm khi đã thấy}*

   i:=1;
2. **While** i <= n **and not** (Tim) **do**

   if V[i]=X **then begin**

```
            Tim: = True;
            K: = i;
            Write(' ',K);
            return
                  end
   Else     i: = i+1;
```

3. **Return**

Số lần phép toán tích cực chỉ phụ thuộc vào chỉ số i mà V[i]=X

Thuận lợi X = V[1] → 1 lần

Xấu X = V[n] → n lần

Vậy $T_{tốt} = O(1)$

$T_{xấu} = O(n)$

Đánh giá thời gian thực hiện giải thuật $T_{tb}(n) = O(n)$

Chương 2

ĐỆ QUY VÀ GIẢI THUẬT ĐỆ QUY

Mục tiêu:

- Người học hiểu được một số khái niệm như đệ quy; giải thuật đệ quy và thủ tục đệ quy. Từ đó người học có thể lấy được các ví dụ về đệ quy và giải thuật đệ quy.

- Chương này cung cấp một số bài toán cơ bản về việc thiết kế giải thuật đệ quy như bài toán tìm n!; bài toán tìm dãy số Fibonaci; bài toán "Tháp Hà Nội;... Từ đó, người học biết cách thiết kế giải thuật đệ quy cho bài toán đệ quy.

- Người học được cung cấp một bài tập mẫu minh hoạ về giải thuật lặp và giải thuật khác để có thể so sánh được về hiệu lực giữa giải thuật đệ quy với một số giải thuật khác như giải thuật lặp;...

I. KHÁI NIỆM VỀ ĐỆ QUY

Một đối tượng gọi 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 rất hay gặp các định nghĩa đệ quy

* Số tự nhiên có thể được định nghĩa như sau:

  a) 1 là số tự nhiên

  b) x là số tự nhiên nếu x-1 là số tự nhiên

* Hàm n giai thừa: n!

  a) 0! = 1

  b) Nếu n > 0 thì n! = n(n-1)!

II. GIẢI THUẬT ĐỆ QUY VÀ THỦ TỤC ĐỆ QUY

Nếu lời giải của một bài toán T được thực hiện bằng lời giải T' có dạng giống như T, thì đó là một lời giải đệ quy. Giải thuật tương ứng với lời giải như vậy gọi là giải thuật đệ quy. Mới nghe có vẻ như "dã tràng xe cát biển Đông",

nhưng vấn đề ở đây là tuy T' có dạng giống như T nhưng theo một nghĩa nào đó T' phải nhỏ hơn T.

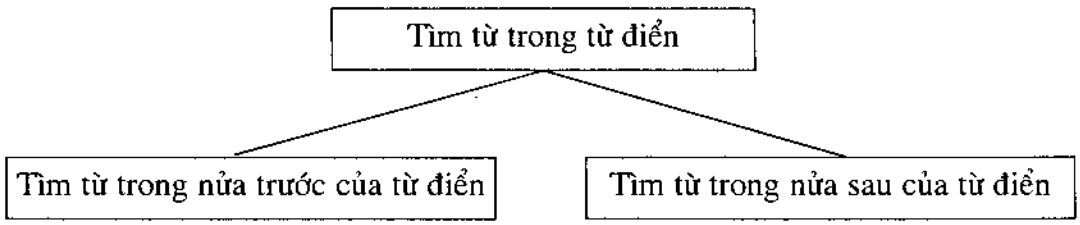
Để minh hoạ ta xét bài toán sau: *Tìm từ trong một quyển từ điển*; sau đây là một giải thuật:

if từ điển là một trang thì tìm từ trong trang ấy
else Begin
Mở từ điển vào trang "giữa"
Xác định xem nửa nào của từ điển chứa từ cần tìm
if từ đó ở nửa trước của từ điển
then tìm từ đó trong nửa trước
else tìm từ đó trong nửa sau
End;

Tất nhiên giải thuật này mới chỉ ở dạng khái quát nhất, có nhiều chỗ cần phải chi tiết hơn, chẳng hạn:

- Tìm từ trong một trang thì làm thế nào
- Làm thế nào để mở từ điển vào trang giữa
- Làm thế nào để biết từ cần tìm nằm ở nửa nào của từ điển,...

Trả lời cho các câu hỏi trên không phải là khó, nhưng ta sẽ không sa vào các chi tiết này mà nên tập trung vào chiến thuật của lời giải. Chiến thuật này có thể hình dung như sau:



Ta có 2 nhận xét sau:

1. Sau mỗi lần tách đôi từ điển thì nửa thích hợp sẽ lại được tìm kiếm bằng chiến thuật đã dùng trước đó.

2. Có một trường đặc biệt sẽ đạt được sau nhiều lần tách đôi, đó là trường hợp khi từ điển chỉ còn duy nhất một trang; lúc đó việc tách đôi chấm dứt và bài toán đủ nhỏ để ta có thể tìm ngay được từ cần thiết (bằng phương pháp tuần tự chẳng hạn). Trường hợp này gọi là suy biến.

Rõ ràng chúng ta đã áp dụng sách lược "chia để trị". Bài toán được tách ra thành bài toán nhỏ hơn và bài toán nhỏ hơn này lại được giả quyết bằng chiến thuật tách nhỏ hơn nữa. Ta sẽ thể hiện giải thuật này dưới dạng một thủ tục.

Procedure SEARCH(dict,word);

{dict được coi là đầu mối để truy nhập vào từ điển đang xét, word chỉ từ cần tìm}

1. **if** chỉ còn là một trang
   then tìm từ word trong trang này.
   else Begin
      Mở từ điển vào trang "giữa"
      Xác định xem nửa nào của từ điển chứa từ word;
      if word nằm ở nửa trước của từ điển
        then call SEARCH (dict1,word)
          else call SEARCH (dict2,word)
      End;

{dict1 và dict2 là đầu mối để truy nhập vào nửa trước và nửa sau của từ điển}

2. **Return**;

* Nếu thủ tục chứa lời gọi đến chính nó, như thủ tục SEARCH ở trên gọi là thủ tục đệ quy trực tiếp.

Thủ tục chứa lời gọi đến thủ tục khác mà ở thủ tục này lại chứa lời gọi đến nó. Trong trường hợp này gọi là đệ quy gián tiếp.

III. THIẾT KẾ GIẢI THUẬT ĐỆ QUY

Khi bài toán đang xét hoặc dữ liệu đang xử lý được định nghĩa dưới dạng đệ quy thì việc thiết kế các giải thuật đệ quy tỏ ra rất hiệu quả.

Ta sẽ minh hoạ nhận xét trên bằng một số bài toán điển hình sau:

1. Bài toán tính n!

Xuất phát từ định nghĩa đệ quy của n! như sau:

$$\text{fact}(n) = \begin{cases} 1 \text{ if } n = 0 \\ n*\text{fact}(n-1) \text{ if } n > 0 \end{cases}$$

Giải thuật đệ quy được viết dưới dạng hàm như sau:

Function fact(n);

1. **if** n=0 **then** fact:=1

   else fact:=n*fact(n-1)

2. **Return**

- Lời gọi chính nó ở đây nằm sau **else**
- Sau mỗi lần gọi n giảm đi 1

Ta thấy 3 đặc điểm của thủ tục đệ quy lại xuất hiện:

1. Có trường hợp suy biến fact(0) = 1

2. Lời gọi chính nó là ở lệnh gán đứng sau **else**

3. Mỗi lần gọi đệ quy đến fact thì giá trị của n giảm đi 1 (ví dụ fact(4) gọi đến fact(3), fact(3) gọi đến fact(2),...)

2. Dãy số Fibonaci

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

1. Giả sử các con thỏ không bao giờ chết

2. Hai tháng sau khi ra đời một cặp thỏ mới sinh ra một cặp thỏ con (một đực, một cái).

3. Khi đã sinh con rồi thì cứ mỗi tháng tiếp 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 đầu vẫn chưa đẻ)

Tháng thứ 3: 2 cặp (đã có thêm một 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)

Tính số cặp thỏ ở tháng n: F(n)

Ta thấy nếu mỗi cặp thỏ ở tháng thứ (n-1) đều sinh con thì:

$$F(n)=2*(n-1)$$

Nhưng thực ra không phải như vậy vì trong các cặp đã có ở tháng thứ (n-1) chỉ có những cặp thỏ ở tháng thứ (n-2) mới sinh con ở tháng thứ n được.

Do đó $F(n) = F(n-2) + F(n-1)$

Vì vậy có thể tính F(n) theo công thức:

$$F(n) = \begin{cases} 1 \text{ if } n <= 2 \\ F(n-2) + F(n-1) \text{ if } n > 2 \end{cases}$$

Dãy số thể hiện các giá trị của n = 1,2,3,... có dạng

1 1 2 3 5 8 13 21 34 55... được gọi là dãy Fibonaci.

Thủ tục đệ quy để thực hiện giải thuật F(n)

Function F(n);

1. **if** n<=2 **then** F:=1

   else f:=F(n-2)+F(n-1)

2. **Return**

Ở đây chỉ khác trường hợp suy biến là có hai giá trị F(1) = 1 F(2) = 1

Chú ý: Đối với hai bài toán đã xét ta thấy việc thiết kế đệ quy khá dễ dàng vì cả hai đều thuộc dạng tính giá trị của hàm mà định nghĩa đệ quy của hàm được xác định khá thuận lợi.

Nhưng không phải khi nào ta cũng gặp thuận lợi như vậy; vấn đề mà chúng ta cần lưu tâm tới khi thiết kế một giải thuật đệ quy là:

1. Có thể định nghĩa một bài toán cùng dạng nhưng "nhỏ" hơn không.

2. Như thế nào là "kích thước" của bài toán được giảm đi sau mỗi lần gọi đệ quy.

3. Trường hợp đặc biệt nào được gọi là suy biến.

Ta xét thêm một bài toán phức tạp hơn:

3. Bài toán "Tháp Hà Nội"

Bài toán được trình bày như sau:

Có n đĩa kích thước khác nhau được xếp chồng lên nhau sao cho đĩa nhỏ nằm trên đĩa lớn (tạo thành hình tháp); cần chuyển n đĩa này (A) đến một vị trí mới (C) theo nguyên tắc:

1. Mỗi bước chỉ chuyển một đĩa

2. Trong quá trình di chuyển không được để đĩa lớn lên trên đĩa bé.

3. Được phép sử dụng một vị trí trung gian (B).

Để đi tới lời giải tổng quát, trước hết ta xét một vài trường hợp đơn giản:

- Trường hợp 1 đĩa: lời giải là chuyển đĩa này từ A sang C

- Trường hợp 2 đĩa: lời giải như sau:

+ Chuyển đĩa đầu từ A sang B

+ Chuyển đĩa thứ 2 từ A sang C

+ Chuyển đĩa đầu từ B sang C

Bây giờ để giải bài toán với n đĩa ta giả sử rằng đã giải được bài toán với n-1 đĩa; lúc đó lời giải tổng quát cho n đĩa như sau:

+ Chuyển (n-1) đĩa trên cùng từ A sang B

+ Chuyển đĩa thứ n từ A sang C

+ Chuyển (n-1) đĩa từ B sang C

Như vậy bài toán "Tháp Hà Nội" với n đĩa được thay thế bằng bài toán tương tự với kích thước nhỏ hơn,... cứ như thế cho tới khi trường hợp suy biến xảy ra.

Các đặc điểm của một giải thuật đệ quy đã được xác định; ta có thể biểu diễn giải thuật đệ quy này như sau:

procedure hanoi(n,A,B,C)

1. **if** n = 1 **then** chuyển đĩa từ A sang C
2. **else begin**

```
        call hanoi(n-1,A,C,B);
        call hanoi(1,A,B,C);
        call hanoi(n-1,B,A,C);
    end;
```

3. **return**

IV. HIỆU LỰC ĐỆ QUY

Đệ quy là một công cụ giải các bài toán, mặc dù bên cạnh nó vẫn có giải thuật khác để thay thế. Khi thay các giải thuật đệ quy bằng các giải thuật không tự gọi chúng gọi là khử đệ quy. Đệ quy cho phép xác định một tập vô hạn các đối tượng bằng một phát biểu hữu hạn. Chẳng hạn:

Giải thuật lặp để tính n! có thể viết:

function fact(n)

1. **if** n = 0 **then** fact:= 1
2. **else begin**

```
        fact := 1;
        for i:=2 to n do fact := i*fact
```

3. **return**

Giải thuật lặp để tính số Fibonaci như sau:

```
function f(n)
1. if n<= 2 then f := 1
2. else begin
        fib1:=1; fib2 :=1;
        for i:= 3 to n do begin
                fibn := fib1 +fib2;
                fib1 := fib2;
                fib2 := fibn;
              end;
3. f := fibn;
4. return;
```

Để kết thúc ta lưu ý rằng trong các trường hợp nếu tránh được đệ quy thì nên tránh. Tại sao?

Vì thuật toán đệ quy rất tiện lợi cài đặt bằng chương trình nhưng khi thực hiện thì tốn rất nhiều thời gian (vì phải tính truy hồi), độ phức tạp tính toán của giải thuật đệ quy thường là hàm mũ. Tuy nhiên trong nhiều trường hợp thuật toán đệ quy là duy nhất hoặc thuận lợi hơn nhiều so với lời giải lặp (ví dụ như trường hợp bài toán "Tháp Hà Nội").

V. MỘT SỐ BÀI TOÁN ÁP DỤNG GIẢI THUẬT ĐỆ QUY

1. Bài toán tính n!

```
PROGRAM giai_thua;
    uses crt;
    Var
       n:longint;
(*-------------*)
    function gt(n:longint):longint;
       Begin
          if n=0 then gt:=1 else gt:=n*gt(n-1);
       End;
(*-------------*)
    BEGIN
       clrscr;
       write('Nhap n= ');readln(n);writeln;
       writeln(n,'! = ',gt(n));
```

```
        readln;
    END.
```

2. Bài toán dãy số Fibonaci

```
PROGRAM day_so_Fibonaci;
    uses crt;
    Var
        n:longint;
(*-------------*)
    function Fn(n:longint):longint;
        Begin
            if n<=2 then Fn:=1 else Fn:=Fn(n-2)+Fn(n-1);
        End;
(*-------------*)
    BEGIN
        clrscr;
        write('Nhap n= ');readln(n);writeln;
        writeln('Day so Fibonaci voi n= ',n,' :');
        writeln('Fn(',n,')= ',Fn(n));
        readln;
    END.
```

3. Bài toán "Tháp Hà Nội"

```
PROGRAM Thap_ha_noi;
    uses crt;
    Var
        n:integer;
        A,B,C:char;
(*-------------*)
    Procedure ThapHN(n:integer;A,B,C:char);
        Begin
             If (n=1) then writeln('Chuyen tu ',A,'
sang ',B)
            else
               Begin
```

```
                    ThapHN(n-1,A,C,B);
                    ThapHN(1,A,B,C);
                    ThapHN(n-1,C,B,A);
                End;
        End;
(*-------------*)
    BEGIN
      clrscr;
      A:='A'; B:='B'; C:='C';
      write('So tang cua thap n=');readln(n);
      ThapHN(n,A,B,C);
      readln;
    END.
```

Chương trình giải bài toán "Tháp Hà Nội" trên được trình bày đơn giản, mô tả đúng lời giải đệ quy bằng lời, dễ hiểu đúng với thuật toán đã trình bày, liệt kê quá trình chuyển chồng đĩa từ cọc A sang cọc B. Sau đây chúng tôi xin trình bày thêm lời giải bài toán "Tháp Hà Nội" bằng đồ họa mô phỏng quá trình chuyển đĩa từ cọc A sang cọc B để chúng ta có thể quan sát quá trình đó bằng hình ảnh đồ hoạ sinh động. *(Chương trình đã được kiểm nghiệm, chạy thử và được trình bày bằng ngôn ngữ lập trình C hết sức ngắn gọn, đơn giản).*

```
//Chương trình mô phỏng bài toán Tháp Hà nội
#include <conio.h>
#include <dos.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>

#define di_chuyen(x,y,k)
tao_khoi(x,y,k);delay(t);xoa_khoi(x,y,k);

int t; //Toc do di chuyen
int a[]={0,3,5,7,9,11,13,15,17,19,21,23,25}; //Do dai
khoi
int mau[]={1,2,3,4,5,6,7,9,10,11,12,13,14,15};//Mau
cua khoi
```

```
int m,mau_nen=0;
int x1=13,y1=22,x2=40,y2=22,x3=67,y3=22;

void main()
  {
        void ve_coc();
        void tao_khoi(int,int,int);
        void xoa_khoi(int,int,int);
        void dung_thap(int,int,int);
        void chuyen_khoi(int,int,int,int,int);
        void chuyen_thap(int,int,int,int,int,int,int);

        clrscr();
        gotoxy(15,1);cprintf("Chuong trinh mo phong
                              bai toan THAP HA NOI");
        gotoxy(23,3);cprintf("Nhap du lieu cho bai toan");
        gotoxy(12,4);cprintf("So tang cua thap m(0<m<13)=");
        cin>>m;
        if(m<1||m>12) exit(1);
        gotoxy(12,5);cprintf("Toc do di chuyen: t(0<t<1000)=")
        ;cin>>t;
        textbackground(mau_nen);clrscr();
        window(1,1,80,2);
        textbackground(BLUE);clrscr();
        textcolor(WHITE);
        gotoxy(34,1);
        cprintf("THAP HA NOI");
        gotoxy(21,2);
        cprintf("(C)Copyright 2004 by Nguyen Thai Ha");
        window(1,23,80,23);
        textbackground(0);clrscr();
        textcolor(YELLOW);
        gotoxy(13,1);cprintf("A");
        gotoxy(40,1);cprintf("B");
        gotoxy(67,1);cprintf("C");
```

```
        dung_thap(x1,y1,m);
        window(1,24,80,24);
        textbackground(0);clrscr();
        textcolor(CYAN);
        gotoxy(27,1);cprintf("Bam Enter de xem di chuyen");
        getch();clrscr();
        chuyen_thap(x1,y1,x2,y2,x3,y3,m);
        window(1,24,80,24);
        textbackground(0);clrscr();
        textcolor(CYAN);
        gotoxy(30,1);cprintf("Bam Enter de ket thuc");
        getch();clrscr();
       window(1,1,80,25);
       textbackground(mau_nen);
       textcolor(WHITE);clrscr();
       system("cls");
   }
void ve_coc()
   {
       int i;
       window(1,1,80,25);
       textbackground(0);
       textcolor(WHITE);
       for(i=0;i<=m;i++)
       {
           gotoxy(x1,y1-i);cprintf("%c",179);
           gotoxy(x2,y2-i);cprintf("%c",179);
           gotoxy(x3,y3-i);cprintf("%c",179);
       }
   }
void tao_khoi(int x,int y,int k)
   {
        int h=a[k]/2;
        window(x-h,y,x+h,y);
        textbackground(mau[k]);
        textcolor(mau[k]);clrscr();ve_coc();
```

```
void xoa_khoi(int x,int y,int k)
   {
        int h=a[k]/2;
        window(x-h,y,x+h,y);
        textbackground(mau_nen);
        textcolor(mau_nen);clrscr();ve_coc();
   }
void dung_thap(int x,int y,int m)
   {
       int i;
       for(i=m;i>=1;--i) tao_khoi(x,y-m+i,i);ve_coc();
   }
void chuyen_khoi(int x1,int y1,int x2,int y2,int k)
   {
        static sl_chuyen=0;
        int x,y;
        window(1,24,80,24);
        textbackground(0);clrscr();
        textcolor(CYAN);
        gotoxy(30,1);cprintf("Bam phim bat ky de dung");
        sl_chuyen++;
        window(1,3,80,3);
        textbackground(0);clrscr();
        textcolor(LIGHTMAGENTA);
        cprintf("Lan di chuyen thu:");
        textcolor(YELLOW);cprintf("%d",sl_chuyen);
        for(y=y1;y>=4;--y)
           {di_chuyen(x1,y,k);}
        if(x1<x2)
        for(x=x1;x<=x2;++x)
           {di_chuyen(x,4,k);}
        else
        for(x=x1;x>=x2;--x)
           {di_chuyen(x,4,k);}
        for(y=4;y<=y2;++y)
           {di_chuyen(x2,y,k);}
        tao_khoi(x2,y2,k);
```

```
        if(kbhit())
        {
              window(1,1,80,25);
              textbackground(mau_nen);
              textcolor(WHITE);
              clrscr();system("cls");exit(1);
        }
    }
void chuyen_thap(int x1,int y1,int x2,int y2,int x3,int y3,int m)
    {
        if(m<1) return;
        else
        if(m==1)
            chuyen_khoi(x1,y1,x2,y2,1);
        else
        {
              chuyen_thap(x1,y1-1,x3,y3,x2,y2,m-1);
              chuyen_khoi(x1,y1,x2,y2,m);
              chuyen_thap(x3,y3,x2,y2-1,x1,y1,m-1);
        }
    }
```

Bài tập

1. Viết một thủ tục đệ quy in ngược một dòng ký tự cho trước.

2. Giải thuật tính ước số chung lớn nhất của hai số nguyên dương p và q (p>q) được mô tả như sau:

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

- Nếu r = 0 thì q là USCLN
- r <> 0 thì gán cho p giá trị của q, gán cho q giá trị của r rồi lặp lại quá trình.

Hãy viết một giải thuật đệ quy và một giải thuật lặp thể hiện hàm đó.

3. Viết một thủ tục đệ quy để in ra hoán vị của n số tự nhiên.

Ví dụ: n = 3 $a_1 = 1$ $a_2 = 2$ $a_3 = 3$

123; 132; 231; 213; 312; 321;

Gợi ý: Hãy để ý nhận xét

1 2 3	**1 3** 2	**2 3** 1
2 1 3	**3 1** 2	**3 2** 1

Chương 3

MẢNG VÀ DANH SÁCH

Mục tiêu:

- Người học hiểu được một số khái niệm về mảng; phần tử của mảng và danh sách.
- Người học biết một số phép tính với danh sách; về địa chỉ của phần tử trong danh sách; địa chỉ của bộ nhớ và cách định địa chỉ của phần tử trong danh sách.
- Người học hiểu được cấu trúc dữ liệu kiểu đơn giản nhất là dùng địa chỉ tính được để lưu trữ và tìm kiếm phần tử là dữ liệu kiểu mảng. Từ đó người học biết các cách truy nhập vào phần tử kiểu mảng.
- Người học biết được cách lưu trữ dữ liệu kế tiếp kiểu danh sách tuyến tính. Người học biết được cách bổ sung hay loại bỏ một phần tử của danh sách.
- Người học hiểu được thế nào là Stack; lưu trữ Stack bằng mảng. Mặt khác người học được cung cấp một số ứng dụng của Stack để hiểu rõ hơn về dữ liệu kiểu Stack.
- Người học hiểu được khái niệm về Queue hay danh sách kiểu hàng đợi. Biết được cách lưu trữ Queue bằng mảng; được cung cấp một số ví dụ về việc bổ sung và loại bỏ một phần tử ra khỏi Queue.

I. CÁC KHÁI NIỆM

Mảng là một tập cố định gồm một số cố định các phần tử. Không có phép bổ sung, loại bỏ phần tử của mảng.

Thường chỉ có các phép tạo lập, tìm kiếm, lưu trữ một phần tử của mảng.

Ngoài giá trị một phần tử của mảng còn được đặc trưng bởi chỉ số, thể hiện thứ tự của các phần tử đó trong mảng.

Véctơ là mảng một chiều, mỗi phần tử a_i ứng với một chỉ số i.

Ma trận là mảng hai chiều, mỗi phần tử a_{ij} ứng với hai chỉ số i,j.

Danh sách hơi khác mảng ở chỗ: gồm một tập có thứ tự nhưng bao gồm một số *biến động* các phần tử. Phép bổ sung hay loại bỏ một phần tử thường xuyên

tác động lên danh sách. Một danh sách mà *lân cận* giữa các phần tử được *hiển thị* ra thì được gọi là danh sách tuyến tính.

Cho ví dụ: tập hợp những người đến phòng công chứng cho ta một hình ảnh của danh sách.

Véctơ chính là trường hợp đặc biệt của danh sách tuyến tính đó là hình ảnh của danh sách tuyến tính xét tại thời điểm nào đấy.

Như vậy danh sách tuyến tính là một danh sách hoặc rỗng (không có phần tử nào) hoặc có dạng $(a_1, a_2, \ldots, a_n)$ với a_i $(1 \le i < n)$ là một dữ liệu nguyên tử. Trong danh sách tuyến tính luôn tồn tại một phần tử đầu a_1, phần tử cuối a_n. Đối với mỗi phần tử a_i bất kì với $1 \le i < n-1$ thì có một phần tử a_{i+1} gọi là phần tử sau a_i, và với $2 < i < n$ thì có một phần tử a_{i-1} gọi là phần tử trước a_i.

a_i được gọi là phần tử thứ i của danh sách tuyến tính, n được gọi là độ dài hoặc kích thước của danh sách, nó có giá trị thay đổi.

Mỗi phần tử trong danh sách thường là một bản ghi gồm một hoặc nhiều trường (fields); đó là thông tin nhỏ nhất có thể tham khảo được trong một ngôn ngữ lập trình; ví dụ danh mục điện thoại là một danh sách tuyến tính, mỗi phần tử của nó ứng với một đối tượng thuê bao. Nó gồm 3 trường:

- Tên thuê bao
- Địa chỉ
- Số điện thoại

Đối với một danh sách ngoài phép bổ sung hay loại bỏ còn có các phép:

- Ghép hai hoặc nhiều danh sách
- Tách một danh sách thành nhiều danh sách
- Sao chép một danh sách
- Cập nhật danh sách
- Sắp xếp các phần tử trong danh sách theo một thứ tự nhất định
- Tìm kiếm trong một danh sách một phần tử mà một trường nào đó có giá trị ấn định.

Các vấn đề mà ta sắp đề cập tới đều liên quan tới bộ nhớ trong; để hình dung ra bộ nhớ trong, ta coi nó như một dãy có thứ tự các từ máy, mỗi từ máy có một địa chỉ; mỗi từ máy thường chiếm từ 8 đến 64 bit, việc tham khảo đến nội dung của nó thông qua địa chỉ.

Có 2 cách định địa chỉ của một phần tử trong danh sách:

* *Cách thứ 1*: Dựa vào đặc tả của dữ liệu cần tìm. Địa chỉ này được gọi là địa chỉ tính được. Cách này thường được sử dụng trong các ngôn ngữ lập trình để tính địa chỉ các phần tử của một vectơ, của ma trận;

* *Cách thứ 2*: Lưu trữ các địa chỉ cần thiết ở một chỗ nào đó trong bộ nhớ. Cần thiết sẽ lấy ra. Địa chỉ này gọi là con trỏ (Pointer) hoặc móc nối (link). Địa chỉ quay lui để quay trở về chỗ gọi ở chương trình chính khi kết thúc chương trình con chính là loại địa chỉ này.

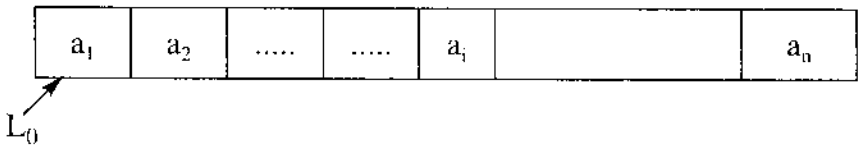
II. CẤU TRÚC LƯU TRỮ CỦA MẢNG

Cấu trúc dữ liệu đơn giản nhất dùng địa chỉ tính được để thực hiện lưu trữ và tìm kiếm phần tử là mảng một chiều (hay véctơ).

Với cách này một số từ máy kế tiếp sẽ được dành ra để lưu giữ các phần tử của mảng (vì vậy mà người ta gọi là lưu trữ kế tiếp).

Giả sử véctơ có n phần tử, mỗi phần tử có thể lưu trữ trong một từ máy. Như vậy phải có n từ máy kế tiếp nhau. Do kích thước của véctơ đã được xác định nên không gian nhớ dành ra cũng được ấn định trước.

Một cách tổng quát, một mảng A có n phần tử nếu mỗi phần tử a_i chiếm c từ máy thì nó sẽ chiếm dụng c*n từ máy liên tiếp như sau:



Địa chỉ a_i sẽ tính được bởi:

$$Loc(a_i) = L_0 + c*(i-1)$$

L_0 được gọi là địa chỉ gốc, là địa chỉ của từ máy đầu tiên trong miền nhớ kế tiếp để lưu trữ mảng vectơ (mà ta gọi là vectơ lưu trữ).

$f(i) = c*(i-1)$ gọi là hàm địa chỉ

Trong một số ngôn ngữ (chẳng hạn như PASCAL) cận dưới chỉ số không nhất thiết phải là 1, mà có thể là một số nguyên b nào đó; trong trường hợp này địa chỉ của a_i sẽ tính như sau:

$$Loc(a_i) = L_0 + c*(i-b)$$

Đối với mảng một chiều, việc tổ chức lưu trữ cũng thực hiện tương tự nghĩa là vẫn bằng một vectơ lưu trữ kế tiếp như trên.

Ví dụ trong FORTRAN một ma trận 3 hàng 4 cột được lưu trữ kế tiếp như sau:

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

Ta thấy cách lưu trữ ở đây là lưu trữ theo cột (Thứ tự ưu tiên cột).

Giả sử mỗi phần tử chiếm một từ máy thì địa chỉ a_{ij} sẽ tính bởi công thức sau: $Loc(a_{ij}) = L_0 + (j-1)*3 + (i-1)$

Một cách tổng quát nếu ma trận có n hàng, m cột thì địa chỉ của a_{ij} sẽ được tính bởi:

$$Log(a_{ij}) = L_0 + (j-1)*n + (i-1)$$

Trong các ngôn ngữ khác (như PASCAL, PL/1,...) cách lưu trữ các phần tử lại theo thứ tự ưu tiên theo hàng. Với ma trận a có n hàng, m cột thì địa chỉ của a_{ij} sẽ được tính như sau:

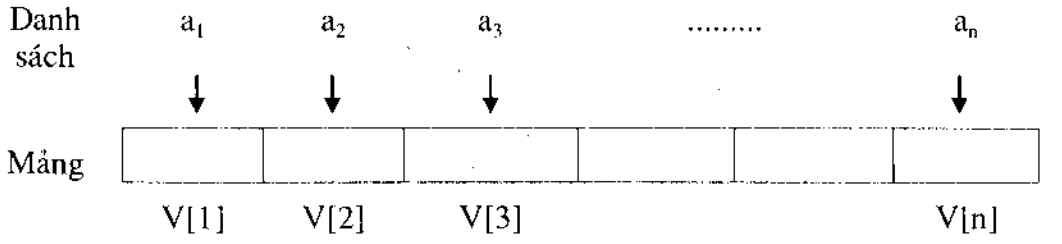
$$Log(a_{ij}) = L_0 + (i-1)*m + (j-1)$$

Nhận xét:

- Khi mảng được lưu trữ kế tiếp thì việc truy nhập vào phần tử của mảng được thực hiện vào địa chỉ tính được, nên tốc độ nhanh và đồng đều với mọi phần tử.
- Có nhiều ô trống → lãng phí bộ nhớ.

III. LƯU TRỮ KẾ TIẾP CỦA DANH SÁCH TUYẾN TÍNH

Vì danh sách là dãy của các mục dữ liệu, cho nên một cách tự nhiên ta có thể dùng mảng một chiều để lưu trữ danh sách tuyến tính nghĩa là có thể dùng 1 véctơ lưu trữ (V_i) với $1 \le i \le n$ để lưu trữ một danh sách tuyến tính ($a_1, a_2,..., a_n$): phần tử a_i được chứa V_i.



Do số phần tử của danh sách tuyến tính thường biến động nghĩa là kích thước n có thể thay đổi. Nếu lấy max(n) mà không dùng hết thì lãng phí bộ nhớ. Ngay cả khi lấy đủ rồi thì phép bổ sung hay loại bỏ phải dồn hoặc dãn danh sách gây lãng phí thời gian. Để nhận thấy điều này ta xét bài toán sau, giả sử chúng ta muốn chèn giá trị 56 sau phần tử 48 trong danh sách mười số nguyên:

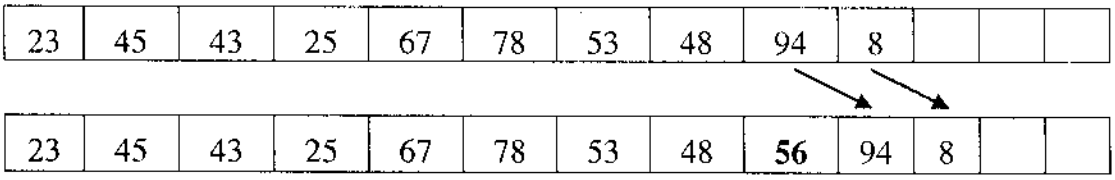
23, 45, 43, 25, 67, 78, 53, 48, 94, 8

Để tạo ra danh sách mới:

23, 45, 43, 25, 67, 78, 53, 48, **56**, 94, 8

Vì định nghĩa của danh sách như là một cấu trúc không có giới hạn về chiều dài cho nên thao tác chèn luôn luôn có thể thực hiện được một cách lý thuyết. Tuy nhiên trong cách cài đặt tuần tự này, độ dài cố định của mảng làm giới hạn chiều dài của danh sách; điều này có nghĩa là trước khi chèn một phần tử mới vào danh sách cần phải kiểm tra xem trong mảng còn chỗ cho nó không.

Điều phức tạp thứ hai xuất hiện từ thực tế là trong cách cài đặt này các phần tử của danh sách được lưu trữ tại các vị trí liên tiếp của mảng; vì vậy để chèn một phần tử mới vào ta cần phải dịch chuyển các phần tử của mảng để tạo chỗ cho nó. Ví dụ đối với bài toán đang xét, các phần tử mảng từ vị trí thứ 8 đến 10 phải dịch sang các vị trí từ 9 đến 11 trước khi phần tử mới được chèn vào vị trí thứ 8:

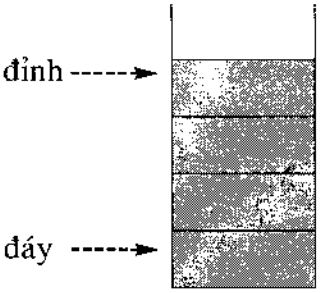


IV. STACK HAY DANH SÁCH KIỂU NGĂN XẾP

1. Định nghĩa

Stack là một kiểu tuyến tính đặc biệt mà phép bổ sung và phép loại bỏ luôn luôn thực hiện ở một đầu gọi là đỉnh (TOP).

Nguyên tắc "Vào sau ra trước" hay danh sách kiểu LIFO (Last In First Out). Stack có thể rỗng hay bao gồm một số phần tử.

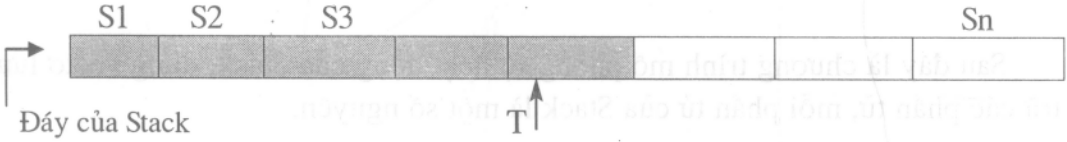


2. Lưu trữ Stack bằng mảng (lưu trữ kế tiếp)

Có thể lưu trữ Stack bởi một véctơ lưu trữ S gồm n phần tử nhớ kế tiếp. Nếu T là địa chỉ của phần tử đỉnh của Stack thì T sẽ có giá trị biến đổi khi Stack hoạt động. (T là một *biến trỏ* - Variable Pointer)

Nếu ta quy ước dùng địa chỉ tương đối (chẳng hạn như chỉ số) thì khi Stack rỗng T=0. Nếu mỗi phần tử của Stack ứng với một từ máy thì khi một phần tử mới bổ sung vào Stack, T sẽ tăng lên 1, khi loại bỏ một phần tử Stack sẽ giảm đi 1.

Cấu trúc lưu trữ kiểu Stack như sau:



* Sau đây là giải thuật bổ sung hay loại bỏ đối với Stack

Procedure Bosung(S,T,X);

{Giải thuật thực hiện bổ sung phần tử X và Stack lưu trữ bởi véctơ X có n phần tử. T là con trỏ trỏ tới đỉnh Stack.}

1. *{Kiểm tra Stack có tràn không}*

```
If T >= n then
        Begin
                Write (' Stack tràn');
                Return
        End;
```

2. {*Chuyển con trỏ*}

   T:=T+1

3. {*Bổ sung phần tử mới X*} S[T]:=X;

4. **Return**.

Function Loaibo(S,T)

{Hàm này thực hiện loại bỏ phần tử ở đỉnh Stack S đang trỏ bởi T. Phần tử bị loại sẽ được thu nhận và đưa ra}

1. {*Xem Stack có cạn không*}
   If T<=0 **then**
      Begin
         Write('Danh sách cạn');
         Return
      End;
2. {*Chuyển con trỏ*}
   T:=T-1;
3. {*Đưa phần tử loại bỏ ra*}
   Loaibo:=S[T+1];
4. **Return**

Sau đây là chương trình mô phỏng sự hoạt động của Stack, dùng vectơ lưu trữ các phần tử, mỗi phần tử của Stack là một số nguyên.

```
Program  _Stack;
uses crt;
const
     nmax=5;
type
     Stack=object
               x:array [1..nmax] of integer;
               n:integer;
               procedure khoitao;
               procedure them;
               procedure xoa(var ptlay:integer);
               procedure xem;
            end;
var
     s:Stack;
     chon,phantu:integer;
(*---------------*)
{Khởi tạo Stack}
procedure Stack.khoitao;
     begin
```

```
        n:=0;
        writeln('Stack da duoc khoi tao');
        readln;
    end;
(*---------------*)
{Thêm một phần tử vào Stack}
procedure Stack.them;
    var
       ptthem:integer;
    begin
        if n=nmax then
           begin
                writeln('Stack tran!!!');
                readln;
           end
        else
           begin
              write('Cho phan tu can them:');readln(ptthem);
              n:=n+1;
              x[n] :=ptthem;
              writeln('Phan tu moi da duoc them vao Stack');
              readln;
           end;
    end;
(*---------------)
{Loại bỏ một phần tử khỏi Stack}
procedure Stack.xoa(var ptlay:integer);
    begin
        if n=0 then
           begin
              writeln('Stack can!!!');
              readln
           end
        else
           begin
```

```
            ptlay:=x[n];
            n:=n-1;
            writeln('Phan tu da duoc lay khoi Stack:',
                                              phantu:8);
            readln;
          end;
    end;
(*--------------*)
{Xem các phần tử của Stack}
Procedure Stack.xem;
    var
       i:integer;
    begin
       if n=0 then writeln('Stack rong !!!');
       for i:=n downto 1 do
          write(x[i]:8);
       writeln;
       readln;
    end;
(*---------------*)
{Tạo menu}
Procedure thongbao;
  begin
       gotoXY(28,5);write('Mo phong su hoat dong cua Stack');
       gotoXY(20,7);write('1. Khoi tao Stack');
       gotoXY(20,8);write('2. Them mot phan tu vao Stack');
       gotoXY(20,9);write('3. Lay mot phan tu tu Stack');
       gotoXY(20,10);write('4. Xem Stack');
       gotoXY(20,11);write('5. Ket thuc');
  end;
(*--------------*)
{Chương trình chính}
BEGIN
```

```
  repeat
    clrscr;
    thongbao;
    gotoXY(20,13);write('Chon cong viec:');readln(chon);
    case chon of
      1:S.khoitao;
      2:S.them;
      3:S.xoa(phantu);
      4:S.xem;
    end;
  until chon=5;
  clrscr;
END.
```

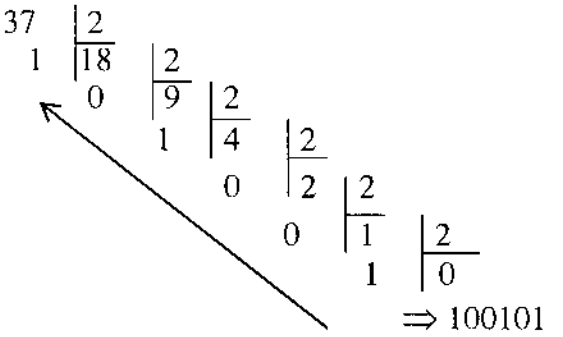
3. Một số ứng dụng của Stack

3.1. Đổi cơ số

Ta biết rằng dữ liệu trong bộ nhớ của máy tính điện tử đều được biểu diễn nhị phân. Điều này có nghĩa là mọi số xuất hiện trong chương trình đều phải chuyển từ dạng thập phân sang dạng nhị phân trước khi thực hiện các phép xử lý. Chẳng hạn số 37 có biểu diễn nhị phân là: 100101

$$1.2^5 + 0.2^4 + 0.2^3 + 1.2^2 + 0.2^1 + 1.2^0 = 37$$

Khi đổi ta chia số đó liên tiếp cho 2 lấy số dư viết theo thứ tự ngược lại.



Đây chính là cơ chế Stack.

* Giải thuật như sau

{*Giải thuật này thực hiện đổi số nguyên cơ số mười sang cơ số 2*}

1. **While** N<>0 **do begin**

    R:= N **mod** 2 {*Tính số dư R trong phép chia N cho 2*}

    Call Bosung(S,T,R) {*Nạp R vào đỉnh Stack*}

    N:= N div 2 {*Thay n bằng thương của phép chia N cho 2*}

    end;

2. **While** S chưa rỗng **do Begin**

    R := Loaibo(S,T) {*Lấy R từ đỉnh Stack S*}

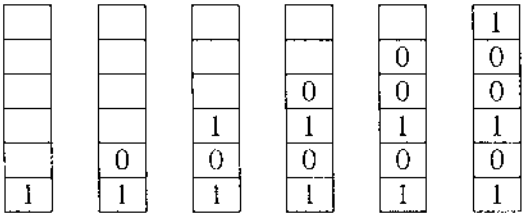
    Write(R);

    End;

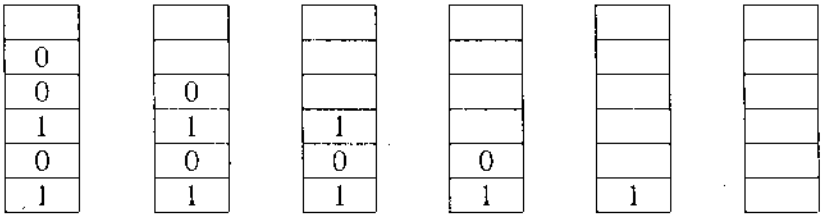
3. **Return**

Ví dụ: $(37)_{10} \rightarrow (101001) = (100101)_2$

Bổ sung:



Loại bỏ:



Sau đây là chương trình đổi cơ số từ số hệ thập phân sang số hệ nhị phân ứng dụng Stack:

```
Program Doi_co_so;
uses crt;
const
```

```
    nmax=16;
type
    Stack=array [1..nmax] of integer;
var
    s:Stack;
    T,phantu,so:integer;
(*---------------*)
procedure khoitao(vart:integer);
    var
        i:integer;
    begin
        for i:=1 to nmax do s[i]:=0;
        t:=0;
    end;
(*-----------------*)
{thủ tục thêm một phần tử vào Stack}
procedure Insert(ptthem:integer;vart:integer);
    begin
            t:=t+1;
            s[t]:=ptthem;
    end;
(*----------------*)
{thủ tục lấy một phần tử ra khỏi stack}
procedure Delete(var ptlay:integer;vart:integer);
    begin
            ptlay:=s[t];
            t:=t-1;
    end;
(*----------------*)
BEGIN
   clrscr;
   write('Nhap vao so can doi:');readln(so);
   khoitao(t);
   while so<>0 do begin
```

```
                    phantu:=so mod 2;
                    so:=so div 2;
                    Insert(phantu,t);
                end;
    writeln;
    write('Doi sang so nhi phan la: ');
    while t<>0 do begin
                    delete(phantu,t);
                    write(phantu);
                end;
    readln;
END.
```

3.2. Định giá biểu thức số học theo ký pháp nghịch đảo Ba Lan

Nhiệm vụ của bộ dịch là tạo ra các chỉ thị máy cần thiết để thực hiện các lệnh chương trình nguồn viết bằng ngôn ngữ lập trình cấp cao.

Ví dụ: X = A*B + C

Bộ dịch máy thực hiện:

1. LOAD A: Tìm giá trị của A lưu giữ trong bộ nhớ và tải vào thanh ghi.
2. MUL B: Tìm giá trị của B và nhân với giá trị đang ở thanh ghi.
3. ADD C: Tìm giá trị của C và cộng với giá trị đang ở thanh ghi.
4. STO X: Đưa giá trị trong thanh ghi lưu vào vị trí X ở bộ nhớ trong.

Trong ngôn ngữ lập trình biểu thức số học được viết như dạng thông thường nghĩa là theo *trung tố* (mỗi ký hiệu của phép toán hai ngôi được đặt giữa hai toán hạng, có thể có thêm dấu ngoặc).

Chẳng hạn 6*(8-3)

Nhà logic học Ba Lan là Lukasiewicz đã đưa ra dạng biểu thức số học theo ký pháp *hậu tố* và *tiền tố*; gọi chung là dạng ký pháp Ba Lan.

Ở dạng hậu tố các toán tử đi sau các toán hạng; lúc đó biểu thức trên sẽ có dạng: **6 8 3-***

Còn ở dạng tiền tố các toán tử đi trước các toán hạng; chẳng hạn biểu thức trên sẽ có dạng: ***6- 8 3**

Ví dụ: (5-2)/(9+(2*3))

Viết theo hậu tố:

5 2-9 2 3*+/

Biểu thức này được đọc từ trái sang phải cho tới khi tìm ra một toán tử; hai toán hạng được đọc cuối cùng, trước toán tử này sẽ được kết hợp với nó.

Trong ví dụ đang xét toán tử "-" được đọc trước tiên 2 toán hạng tương ứng với nó là 5 2 kết quả là 3. Thay vào có biểu thức rút gọn:

3 9 2 3*+/

Lại đọc từ trái qua phải, toán tử tiếp theo là "*" và 2 toán hạng của nó là 2, 3, thực hiện phép toán ta có dạng rút gọn:

3 9 6 +/

Tiếp tục ta thu được:

3 15 /

Cuối cùng ta có: 3/15

Phương pháp định giá trị hậu tố như trên đòi hỏi lưu trữ các toán hạng cho tới khi một toán tử đọc từ trái sang phải. Tại thời điểm này hai toán hạng cuối cùng phải được tìm ra và kết hợp với toán tử này. Như vậy đã xuất hiện cơ chế hoạt động "vào sau ra trước"; nghĩa là ta sẽ sử dụng Stack để lưu trữ các toán hạng; cứ mỗi lần đọc được toán tử thì hai giá trị sẽ được lấy ra từ Stack để kết hợp với toán tử và kết quả lại được đẩy vào Stack; giải thuật sau sẽ thực hiện các ý đồ trên:

```
Program DINH_GIA;
   {Giải thuật này sử dụng 1 Stack S, Trỏ bởi T, lúc đầu T=0}
1. Repeat
   Đọc phần tử X tiếp theo trong biểu thức;
   if X là toán hạng then Call Bo sung(S,T,X)
   else Begin
            Y:=Loaibo(S,T)
            Z:=Loaibo(S,T)
            Tác động toán tử X vào Y, Z gán kết quả cho W
            Bo sung(S,T,W)
        End;
until gặp dấu kết thúc biểu thức;
2. R:=Loaibo(S,T);
```

Write(R);

3. **Return**

V. QUEUE HAY DANH SÁCH KIỂU HÀNG ĐỢI

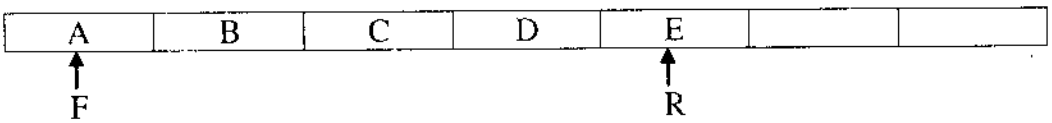
1. Định nghĩa

Khác với Stack, Queue là kiểu danh sách tuyến tính, mà phép bổ sung được thực hiện ở một đầu gọi là lối sau và phép loại bỏ thực hiện ở một đầu khác gọi là lối trước.

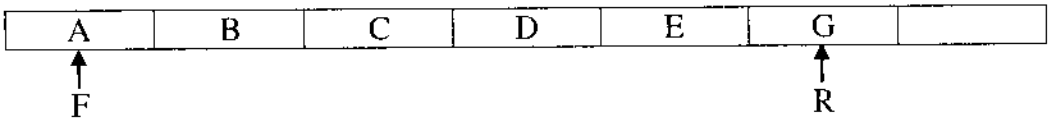
Như vậy cơ cấu của Queue giống như một hàng đợi vào một đầu ra ở đầu khác. Vào trước ra trước. Vì vậy Queue còn được gọi là danh sách kiểu FIFO (First In First Out).

2. Lưu trữ Queue bằng mảng (lưu trữ kế tiếp)

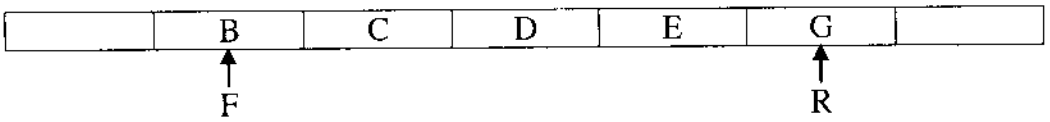
Có thể dùng một véctơ Q có n phần tử làm cấu trúc của Queue. Có thể truy nhập vào Queue ta phải dùng hai biến trỏ: R trỏ tới lối sau và F trỏ tới lối trước. Khi Queue rỗng thì R=F=0. Nếu mỗi phần tử của Queue được lưu trữ cùng một từ máy thì khi bổ sung một phần tử vào Queue R sẽ tăng lên 1, còn khi loại bỏ một phần tử khỏi Queue thì F sẽ tăng lên 1.



Các phần tử ban đầu có trong Queue



Sau khi bổ sung phần tử G



Sau khi loại bỏ phần tử A

Có thể xuất hiện tình huống các phần tử của Queue sẽ di chuyển khắp không gian nhớ khi thực hiện phép bổ sung và loại bỏ. Chẳng hạn cứ liên tiếp thực hiện một phép bổ sung rồi loại bỏ. Do đó người ta phải khắc phục bằng cách coi không gian nhớ tổ chức theo kiểu vòng tròn và với véctơ lưu trữ Q thì Q[1] đứng sau Q[n].

Giải thuật bổ sung và loại bỏ như sau:

Procedure Bosung(F,R,Q,n,XX);

{Cho F và R là hai con trỏ lần lượt trỏ vào lối trước và lối sau của một Queue được lưu trữ kiểu vectơ gồm n phần tử theo kiểu vòng tròn. Giải thuật này nhằm bổ sung một phần tử mới vào lối sau Queue.

Lúc đầu Queue rỗng: F=R=0}

1. {Chỉnh lại con trỏ R}

If R=n **then** R:=1

else R:=R+1;

2. {*Kiểm tra tràn*}

If F=R **then Begin**

Write('Tràn');

Return;

End;

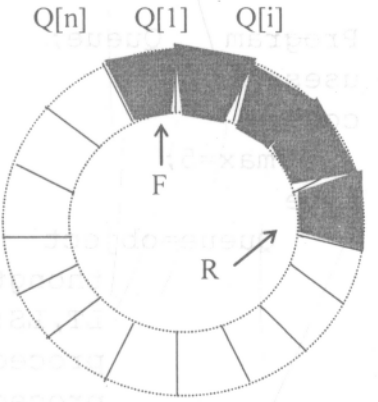
3. {*Bổ sung X vào*}

Q[R]:=X;

4. {*Điều chỉnh con trỏ F khi bổ sung lần đầu*}

If F=0 **then** F:=1;

5. **Return.**



Hình 3.1

Function Loaibo(F,R,Q,n);

1. {*Kiểm tra Cạn*}

If F=0 **then Begin**

Write(' Cạn ');

Return(0);

End;

2. {*Loại bỏ*}

Y:= Q[F];

3. {*Xử lý trường hợp Queue trở thành rỗng sau phép loại bỏ*}

If F=R **then** {*Queue chỉ có một phần tử*}
Begin
F:=R:=0;
Return(Y);
End;

4. {*Chỉnh lại con trỏ F sau phép loại bỏ*}
if F=n **then** F:=1
else F:=F+1;
5. **Return**(Y).

Sau đây là chương trình mô phỏng sự làm việc của Queue nối vòng sử dụng vectơ lưu trữ các phần tử, mỗi phần tử là một số thực.

```
Program  _Queue;
uses crt;
const
     max=5;
type
     Queue=object
               thongtin:array[1..max] of real;
               LT,LS:integer;
               procedure khoitao;
               procedure bosung;
               procedure loaibo(var infolay:real);
               procedure xem;
             end;
var
   Q:Queue;
   chon:byte;
   info:real;
(*----------------*)
{Khởi tạo Queue}
procedure Queue.khoitao;
   begin
     LT:=0;LS:=0;
     writeln('Queue da duoc khoi tao!!!');
```

```
      readln;
   end;
(*-----------------*)
{Bổ xung một phần tử vào trong Queue}
procedure Queue.bosung;
   begin
      write('Cho noi dung can bo sung: ');readln(info);
      if LS=max then LS:=1 else LS:=LS+1;
      if LT=LS then begin
                        write('Queue tran!!!');
                        readln;LS:=LS-1;
                        exit;
                    end;
      thongtin[LS]:=info;
      write('Phan tu moi da duoc bo sung!');
      if LT=0 then LT:=1;
      readln;
   end;
(*-----------------*)
{Loại bỏ một phần tử ra khỏi Queue}
Procedure Queue.loaibo(var infolay:real);
   begin
      if LT=0 then begin                {Queue rỗng}
                      write('Queue can!!!');
                      readln;
                      exit;
                   end;
         infolay:=thongtin[LT];
      if LT=LS then {co 1 pt sau khi loai bo thi rong}
         begin
            LT:=0;LS:=0;
            writeln('Da loai bo 1 phan tu!');
            writeln('Phan tu bi loai bo:',infolay:3:2);
            readln;exit;
         end;
      if LT=max then LT:=1 else LT:=LT+1;
      writeln('Da loai bo 1 phan tu!');
```

```
      writeln('Phan tu bi loai bo:',infolay:3:2);
      readln;
   end;
(*-----------------*)
{Xem các phần tử trong Queue}
Procedure Queue.xem;
   var
      i:integer;
   begin
      if LT=0 then begin
                        write('Queue rong!!!');
                        readln;
                        exit;
                   end;
      writeln('Cac phan tu cua Queue:');writeln;
      if LS<LT then begin
                       for i:=LT to max do
                          write(thongtin[i]:6:2);
                       for i:=1 to LS do
                          write(thongtin[i]:6:2);
                    end
               else begin
                       for i:=LT to LS do
                           write(thongtin[i]:6:2);
                    end;
      readln;
   end;
(*-----------------*)
{Tao menu}
Procedure menu;
  begin
    gotoXY(28,5);write('Queue noi vong');
    gotoXY(20,7);write('1. Khoi tao Queue');
    gotoXY(20,8);write('2. Them mot phan tu');
    gotoXY(20,9);write('3. Xem cac phan tu cua Queue');
    gotoXY(20,10);write('4. Loai bo mot phan tu');
    gotoXY(20,11);write('5. Ket thuc');
```

```
  end;
(*-----------------*)
{Chương trình chính}
BEGIN
  repeat
    clrscr;
    menu;
    gotoXY(20,13);write('Chon cong viec: ');readln(chon);
    case chon of
      1:Q.khoitao;
      2:Q.bosung;
      3:Q.xem;
      4:Q.loaibo(info);
    end;
  until chon=5;
  clrscr;
END.
```

Bài tập

1. Cho ma trận A= (A_{ij}) với $1 \le i \le 7$, $1 \le j \le 6$. Mỗi phần tử của ma trận chiếm 2 từ máy. Hãy viết công thức tính địa chỉ phần tử (A_{ij}) ứng với cách lưu trữ.

a) Theo thứ tự ưu tiên cột.

b) Theo thứ tự ưu tiên hàng.

2. Giả sử 4 từ máy mới đủ lưu trữ một phần tử của ma trận trong PASCAL. Biết rằng một ma trận A được khai báo kiểu:

array [1..8,1..3] **of integer**

Và được lưu trữ trong một miền nhớ kế tiếp bắt đầu từ từ máy có địa chỉ là 2000. Hãy tính Loc(A[4,2]).

3. Hãy lập giải thuật bổ sung và loại bỏ đối với hai Stack hoạt động theo hướng ngược chiều nhau trên một miền nhớ cho phép là một vectơ V có n phần tử (đáy của Stack 1 sẽ là V[1], còn đáy của Stack 2 sẽ là V[n]). Giả sử mỗi một phần tử của Stack ứng với một phần tử của V.

4. Hãy viết biểu thức sau đây dưới dạng hậu tố:

$$\frac{(A*B)}{(C+D)}$$

Dựng hình ảnh của Stack được sử dụng để định giá trị biểu thức hậu tố này ứng với A = 20, B = 4, C = 9, D = 7.

Chương 4

DANH SÁCH MÓC NỐI

Mục tiêu:

- Người học hiểu được thế nào là danh sách nối đơn và cấu tạo của một phần tử trong danh sách nối đơn. Người học biết được mối liên kết giữa các phần tử trong danh sách nối đơn.

- Người học được cung cấp một số phép toán; một số giải thuật để bổ sung một nút mới vào trong danh sách; loại bỏ một nút ra khỏi danh sách; ghép hai danh sách nối đơn với nhau.

- Người học được bổ túc thêm kiến thức về ngôn ngữ lập trình Pascal để viết chương trình cho giải thuật.

- Người học hiểu được thế nào là danh sách nối vòng; sự liên kết giữa các phần tử trong danh sách nối vòng và giải thuật để duyệt danh sách nối vòng.

- Người học hiểu được cấu trúc của danh sách nối kép; cấu trúc của các trường của các nút trong danh sách nối kép. Ngoài ra, người học biết được một số giải thuật liên quan đến danh sách nối kép như giải thuật chèn một phần tử vào trước một giá trị; loại bỏ một giá trị;... đối với danh sách nối kép.

- Ngoài ra chương này còn cung cấp một số ví dụ và bài tập mẫu cho người học củng cố kiến thức và rèn luyện về các giải thuật liên quan đến các loại danh sách.

Lưu trữ kế tiếp với danh sách tuyến tính bộc lộ nhiều nhược điểm trong trường hợp thường xuyên các phép bổ sung hoặc loại bỏ phần tử. Trường hợp xử lý đồng thời nhiều danh sách hoặc trường hợp các ma trận thưa.

Việc sử dụng con trỏ hoặc mối nối để tổ chức danh sách tuyến tính mà ta gọi là danh sách móc nối (liên kết) nhằm khắc phục một số nhược điểm đó.

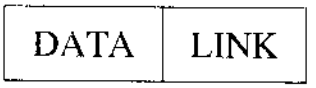
Danh sách móc nối là một danh sách mà các phần tử (còn gọi là mục tin hay nút) được nối kết với nhau nhờ vào vùng liên kết của chúng.

I. DANH SÁCH NỐI ĐƠN

1. Nguyên tắc

Mỗi phần tử của danh sách được lưu giữ trong một phần tử nhớ gọi là nút. Mỗi nút gồm một số từ máy kế tiếp. Các nút này có thể nằm ở bất kỳ vị trí nào trong bộ nhớ.

Mỗi nút gồm hai phần: Phần thông tin và phần chứa địa chỉ của phần tử đứng sau trong danh sách.



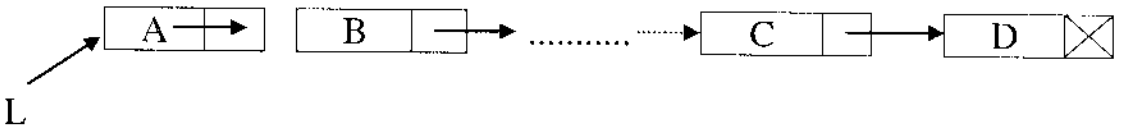
Trường DATA (DATA) chứa thông tin của phần tử

Trường LINK chứa địa chỉ của phần tử tiếp theo

Riêng nút cuối cùng thì không có nút đứng sau nó nên mối nối ở nút này phải là một "địa chỉ đặc biệt" chỉ dùng để đánh dấu nút kết thúc danh sách chứ không như các địa chỉ ở các nút khác, ta gọi "mối nối không" và ký hiệu là Null.

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

Nếu dùng mũi tên chỉ mối nối, ta sẽ có một hình ảnh của một danh sách nối đơn như sau:



Dấu  để chỉ mối nối không (còn gọi là Null).

Nếu danh sách rỗng thì L = Null.

Để tổ chức thành danh sách móc nối thì cần có các điều kiện sau:

1) Tồn tại những phương tiện để chia bộ nhớ thành các nút và ở mỗi nút có thể truy nhập vào từng trường.

2) Tồn tại một cơ chế để xác định được một nút đang được sử dụng (gọi là nút bận) và không được sử dụng (gọi là nút trống).

3) Tồn tại một cơ chế "Ngân hàng nút trống" để cung cấp nút trống khi có yêu cầu sử dụng và thu hồi lại các nút đó nếu không cần thiết.

Ta xét đến vấn đề này sau; trước mắt để tiện trình bày ta gọi cơ chế này là "Danh sách chỗ trống".

Ta quy ước:

X <— Laytra

Là phép lấy một nút có địa chỉ X từ "Danh sách chỗ trống" ra để sử dụng.

X —> Laytra

Là phép trả lại một nút có địa chỉ X về danh sách chỗ trống (còn gọi là phép thu hồi nút X).

2. Một số phép toán

Các giải thuật sau đây thể hiện một số phép toán thường được tác động vào danh sách nối đơn.

Một nút có địa chỉ là P (được trỏ bởi P) thì DATA(P) chỉ trường DATA của nút ấy và LINK(P) chỉ trường LINK của nó.

2.1. Bổ sung một nút mới vào danh sách nối đơn

Procedure CHEN(L,M,X);

{*L là con trỏ trỏ tới nút đầu tiên của danh sách nối đơn, M là con trỏ trỏ tới nút đang có trong danh sách. Giải thuật này thực hiện bổ sung vào sau nút đang trỏ bởi M một nút mới mà trường DATA của nó có giá trị lấy từ ô nhớ có địa chỉ là X*}

1. Tạo một nút mới

   NEW <— Laytra;

   DATA(NEW):=X;

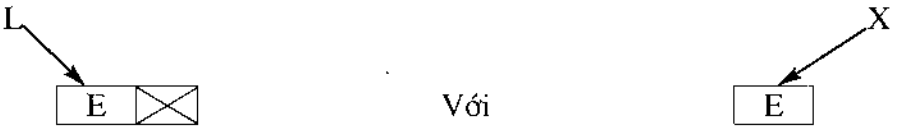
2. {*Thực hiện bổ sung: Nếu danh sách rỗng thì bổ sung một nút mới vào thành nút đầu tiên, nếu danh sách không rỗng thì nắm lấy M và bổ sung nút mới vào sau nút đó*}

```
If L = null then Begin
        L:=NEW;
        LINK(NEW):=Null;
        End
else Begin
        LINK(NEW):=LINK(M);
        LINK(M):=NEW;
        End;
```

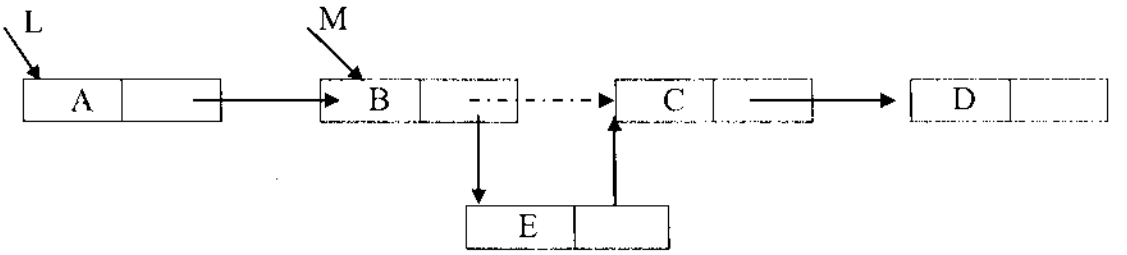
3. **Return.**

Minh hoạ:

a) Nếu L = Null, sau phép bổ sung ta có



b) Nếu L <> null



2.2. Loại bỏ một nút ra khỏi danh sách nối đơn

Program XOA(L,M);

{Cho danh sách nối đơn trỏ bởi L giải thuật này thực hiện loại bỏ nút trỏ bởi M ra khỏi danh sách}

1. *{Trường hợp danh sách rỗng}*

```
    If L=null then Begin Write('Danh sách rỗng');
    Return
end;
```

2. *{Trường hợp nút trỏ bởi M là nút đầu tiên của danh sách}*

```
    If M=L then Begin
        L:=LINK(M);
        M —> Laytra;
        Return;
    End;
```

3. *{Tìm đến nút đứng trước nút trỏ bởi M}*

```
    P:=L;
    While LINK(P) <>M do P:=LINK(P);
```

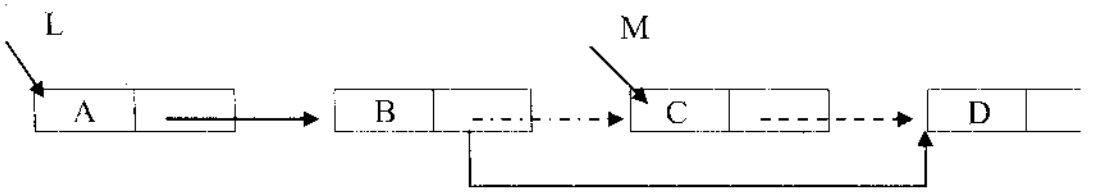
4. *{Loại nút trỏ bởi M}*

```
    LINK(P):=LINK(M);
```

5. {*Đưa nút bị loại về danh sách chỗ trống*}

M —> Laytra

6. **Return**.



2.3. Ghép hai danh sách nối đơn

Program GHEP(P,Q);

{*Cho hai danh sách nối đơn trỏ bởi P và Q. Ghép hai danh sách đó thành một danh sách mới và cho P trỏ tới nó*}

1. {*Trường hợp danh sách trỏ bởi Q rỗng*}

   If Q=Null **then Return**;

2. {*Trường hợp danh sách trỏ bởi P rỗng*}

   If P=Null **then Begin**

   P:=Q; **Return**; **End**;

3. {*Tìm đến nút cuối danh sách*}

P1:=P;

While Link(P1) <> Null **do** P1:=Link(P1)

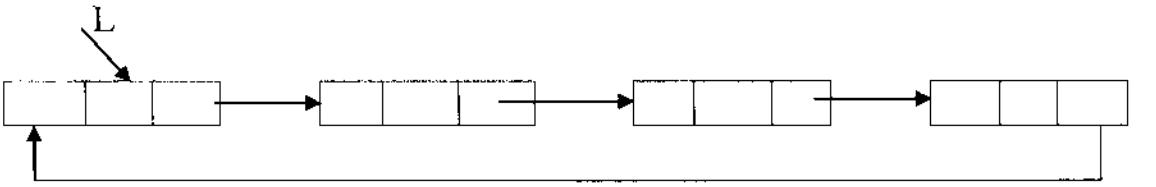
4. {*Ghép*}

   Link(P1):=Q;

5. **Return**.

II. DANH SÁCH NỐI VÒNG

Danh sách nối vòng là danh sách tuyến tính trong đó mối nối của phần tử cuối cùng của danh sách lại trỏ tới phần tử đầu tiên; điều này có nghĩa là sẽ không có phần tử đầu tiên và phần tử cuối cùng. Chúng ta có thể duyệt danh sách từ bất kỳ một nút nào cũng được.



Kiểu danh sách này hay được sử dụng trong các hệ quản trị cơ sở dữ liệu và lập trình hệ thống. Cấu trúc danh sách cho phép liên kết các phần tử có cùng tính chất. Vì vậy chỉ cần biết một phần tử có tính chất nào đó, ta có thể liệt kê hết các phần tử có cùng tính chất đó bằng cách duyệt trên danh sách vòng.

```
procedure duyet(L);
begin
   P := L;
   ok := false;
   while không kết thúc do
      begin
         xử_lý(p);
         p :=LINK(p);
         ok :=  P := L;
      end;
end.
```

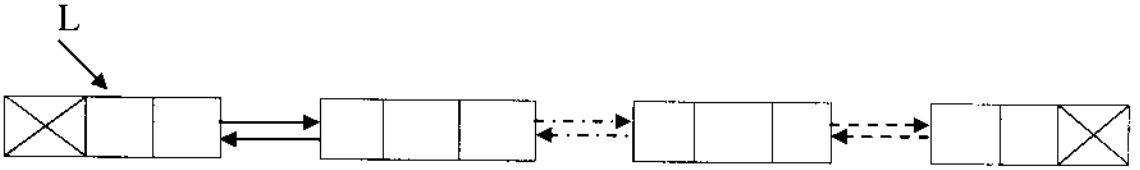
III. DANH SÁCH NỐI KÉP

Trong trường hợp dùng mảng để biểu diễn một danh sách, ta có thể duyệt mảng theo cả hai chiều; trong trường hợp danh sách nối đơn (như đã trình bày ở trên) ta chỉ có thể duyệt từ trái qua phải.

Nếu ta muốn duyệt danh sách từ phải qua trái, cần phải bổ sung thêm một con trỏ nữa, cho phép truy nhập tới phần tử đứng trước; ta sẽ gọi loại danh sách này là danh sách nối kép. Mỗi nút của danh sách sẽ có cấu trúc như sau:

Trong đó: - Ltro là con trỏ trái, dùng để chỉ đến nút đứng trước
- Rtro là con trỏ phải, dùng để chỉ đến nút đứng sau
- DATA chứa dữ liệu của nút.

Hình ảnh của danh sách kép có thể mô tả như sau:



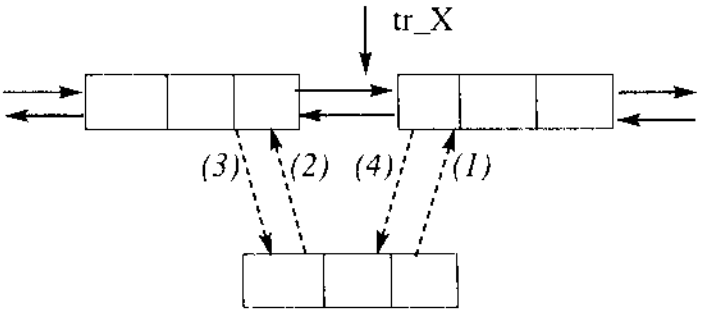
Sau đây ta sẽ nêu ra hai giải thuật cập nhật một danh sách kép; giả sử rằng đã có sẵn hàm troX(ds, X) cho phép truy nhập tới phần tử chứa lần xuất hiện đầu tiên của X.

1. Chèn một nút vào trước nút đã có trong danh sách

Giả sử ta cần chèn một phần tử **f_tu** vào trước vị trí xuất hiện đầu tiên của một giá trị X, ta sử dụng một biến ok để biết việc chèn có thể thực hiện được hay không.

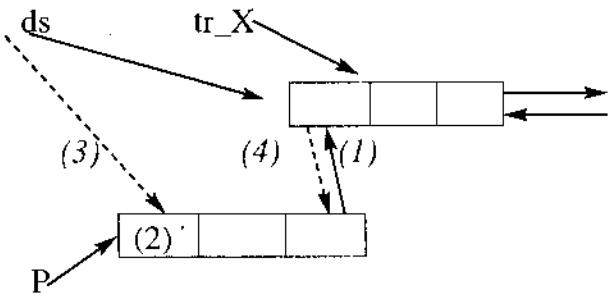
Giả thiết rằng con trỏ tr_X mang địa chỉ của phần tử chứa lần xuất hiện đầu tiên của X, còn p mang địa chỉ của phần tử cần chèn.

Trong trường hợp tổng quát, ta có tình huống sau:



Cần phải thực hiện sửa đổi các con trỏ (1), (2), (3), (4) theo đúng thứ tự đó.

Trường hợp chèn vào đầu ta có:



Ta cần điều chỉnh lại các liên kết (1), (2), (3), (4).

Giải thuật được trình bày như sau:

```
procedure chenX(ds; f_tu; X; ok)
begin
  tr_X := troX(ds,X);
  ok:= false;
  if tr_X <> null then
   begin
      ok := true;
      p ← Laytra;
      DATA(p) := f_tu;
      LINK(p) := tr_X;    {(1)}
      Ltro(p) := Ltro(tr_X);    {(2)}
      If tr_X = ds then {chèn vào đầu danh sách}
           Ds := p
        Else              {trường hợp tổng quát}
          Ltro(tr_X)(Rtro) := p;        {(3)}
       Ltro(Ltr_X) := p;                {(4)}
   end;
   end
```

Một cách tương tự bạn đọc có thể tìm ra giải thuật cho các trường hợp sau:

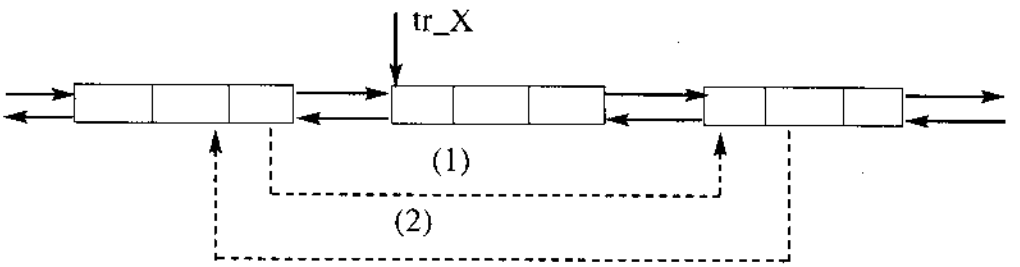
- Giải thuật chèn một phần tử vào vị trí thứ k.
- Chèn một phần tử sau mỗi lần xuất hiện của giá trị X.

2. Giải thuật loại bỏ một phần tử

Ta cần loại bỏ, chẳng hạn lần xuất hiện đầu tiên của một giá trị X trong danh sách kép; ta có ba trường hợp:

- Bỏ phần tử đầu.
- Bỏ phần tử cuối
- Trường hợp tổng quát.

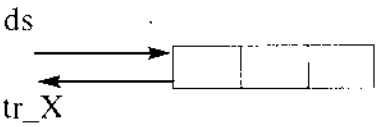
Trong trường hợp tổng quát ta có:



Ta thực hiện các điều chỉnh (1), (2) theo thứ tự bất kỳ.

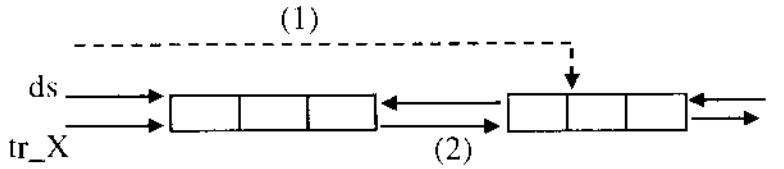
Trong trường hợp bỏ phần tử đầu, ta lại có hai trường hợp nhỏ:

1. Danh sách có đúng một phần tử, sau khi loại bỏ thì danh sách rỗng



Ta chỉ cần thực hiện ds: = LINK(ds);

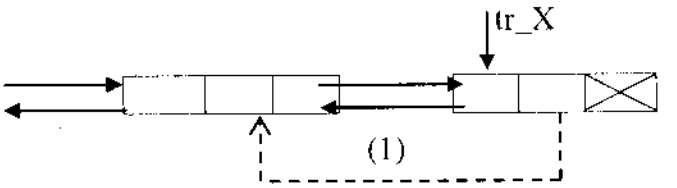
2. Danh sách có hơn một phần tử



Ta chỉ cần điều chỉnh (1) và (2).

Nếu cần bỏ phần tử cuối cùng, ta cũng có hai trường hợp nhỏ:

1. Danh sách chỉ có một phần tử (tương tự như trường hợp vừa xét)
2. Danh sách có hơn một phần tử:



Ta chỉ cần thực hiện (1)

Giải thuật như sau:

```
procedure loai(ds; X; ok);
begin
   ok := false;
   tr_X := troX(ds,X);
   if tr_X <> null then
      begin
         ok := true;
```

```
        if tr_X = ds  then          {bỏ phần tử đầu}
            ds := Rtro(ds)
            else                    {bỏ bên trong}
              Rto(Ltro(tr_X)) := R_tro(tr_X)      {(1)}
        if  Rtro(tr_X) <> null  then    {không phải là phần tử cuối}
           Ltr(Rtro(tr_X)) := Ltro(tr_X)
    end;
end.
```

Độc giả hoàn toàn có thể trình bày giải thuật:

1. Loại bỏ phần tử thứ k
2. Loại bỏ tất cả các lần xuất hiện của giá trị X.

3. Ví dụ áp dụng

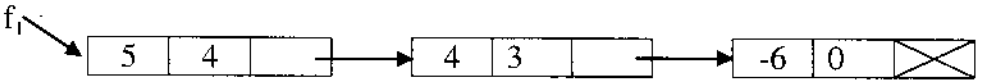
Xét bài toán cộng hai đa thức của x được tổ chức dưới dạng danh sách liên kết.

3.1. Biểu diễn đa thức

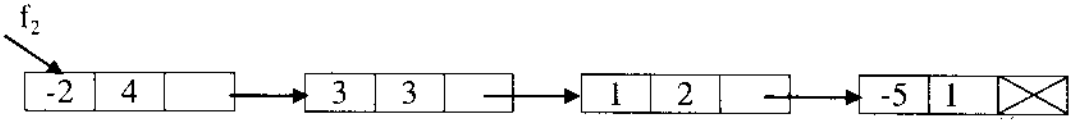
Đa thức sẽ được biểu diễn dưới dạng danh sách nối đơn, với mỗi nút có quy cách như sau:

Ở đây trường HESO chứa hệ số khác 0 của một số hạng trong đa thức, trường MU chứa số mũ tương ứng trong trường LINK chứa mối nối đến nút tiếp theo.

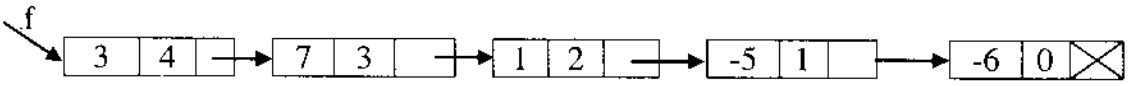
Ví dụ đa thức $f_1(x) = 5x^4 + 4x^3 - 6$ sẽ được biểu diễn bởi:



Còn đa thức $f_2(x) = -2x^4 + 3x^3 + x^2 - 5x$ được biểu diễn bởi:



$f(x) = f_1(x) + f_2(x) = 3x^4 + 7x^3 + x^2 - 5x - 6$ sẽ được biểu diễn như sau:



3.2. Giải thuật

Dùng hai con trỏ p và q duyệt hai danh sách $f(x_1)$, $f(x_2)$ sẽ xuất hiện các tình huống sau:

a) mũ (p) = mũ (q) ta cộng hai giá trị HESO của hai nút đó, nếu kết quả là khác 0 thì phải tạo ra nút mới thể hiện số hạng tổng đó và gắn vào f(x).

b) Nếu mũ (p) <> mũ (q)

if mũ (p) > mũ (q) sao chép nút p và gắn vào danh sách của f.

if mũ (p) < mũ (q) sao chép nút q và gắn vào danh sách của f.

c) Nếu một danh sách kết thúc trước, phần còn lại của danh sách kia sẽ được cất vào f(x).

Mỗi lần một nút mới được tạo ra đều phải gắn vào "đuôi" của f. Như vậy phải thường xuyên nắm được "đuôi" của f bằng một con trỏ d. Công việc này lặp đi lặp lại cho nên cần được thể hiện bằng một thủ tục:

```
Procedure Catdathuc(H,M,d);  {Gắn vào sau nút trỏ bởi d}
    New <— laytra;
    Heso(New):=H;
    mu(New):=M;
    Link(d):= new;
    d:=New; {nút mới trở thành nút đuôi}
    Return.
```

Procedure Cong_da_thuc(f_1,f_2,f);

1. p:= f_1; q:=f_2;
2. f <— laytra;

d:= f *{Lấy nút mới làm "nút đuôi giả" để sử dụng thủ tục* ***catdathuc*** *sau sẽ bỏ đi}*

```
3. While p <> nil and q <> nil do
Case
      mũ(p)=mu(q):    x:=HESO(p) + HESO(q);
            if x<>0 then Call Catdathuc(x,mu(p),d);
            p:=link(p);  q:=link(q);
      mũ(p)<mũ(q): Call Catdathuc(HESO(q),mũ(q),d);
            q:=link(q);
      Else:   Call Catdathuc(HESO(p),mũ(p),d);
            p:=link(p);
```

```
End Case;
4. {Danh sách ứng với f2(x) đã hết}
While p <> nil do Begin
        Call Catdathuc(HESO(p), mũ(p),d);
        p:=link(p);
End;
5. {Danh sách ứng với f1(x) đã hết}
While q <> nil do begin
        Call Catdathuc(HESO(q), mũ(q),d);
        q:=link(q);
End;
6. {Kết thúc danh sách f}
Link(d):=nil;
7. {Loại bỏ nút đuôi giả}
t:=f; f:=link(f); t —> laytra;
8. Return.
```

IV. STACK VÀ QUEUE MÓC NỐI

Với danh sách nối đơn trỏ bởi L thì L coi như trỏ tới đỉnh của Stack. Bổ sung một nút vào Stack chính là bổ sung một nút vào thành nút đầu tiên của danh sách. Loại bỏ một nút khỏi Stack chính là loại bỏ nút đầu tiên trong danh sách trỏ bởi L. Với Stack móc nối không phải kiểm tra hiện tượng tràn như Stack kế tiếp vì kích thước của nó chỉ phụ thuộc vào kích thước của bộ nhớ toàn phần.

Đối với Queue thì loại bỏ một đầu và bổ sung một đầu khác. Nếu coi danh sách nối đơn như một Queue và coi L trỏ tới lối trước thì việc loại bỏ một nút như Stack. Nhưng bổ sung một nút thì thực hiện vào "đuôi" nghĩa là phải tìm đến nút cuối cùng. Vậy cần tìm thêm một con trỏ nữa (P) trỏ vào nút cuối cùng.

V. MỘT SỐ CHƯƠNG TRÌNH MINH HOẠ

1. Chương trình chèn thêm một nút vào danh sách nối đơn

Viết chương trình chèn một nút mới vào trong danh sách nối đơn. Với các chức năng:

- Khởi tạo danh sách
- Chèn vào đầu danh sách
- Chèn vào cuối danh sách

- Chèn vào vị trí bất kỳ nhập vào từ bàn phím
- Xem danh sách

Chương trình được tổ chức dưới dạng menu chức năng

```
(*-------Khai báo-------*)
uses crt;
type
        tro=^nut;
        Nut=record
                info:integer;
                link:tro;
            end;
   var
     L:tro;
     X,k:integer;
     chon:integer;

(*---thủ tục khởi tạo danh sách---*)

   procedure khoitao(var l:tro);
     begin
        l:=nil;
        writeln('Danh sach da duoc khoi tao!!!');
        writeln('Bam phim Enter de tiep tuc!!!');
     end;

(*----thủ tục chèn phần tử vào đầu danh sách---*)

   procedure insert_dau(var l:tro;X:integer);
     var
       moi:tro;
     begin
       new(moi);
       moi^.info:=X;
       if l=nil then begin
```

```
                        moi^.link:=l;
                        l:=moi;
                    end
        else
            begin
                moi^.link:=l;
                l:=moi;
            end;
    end;

(*---thủ tục chèn phần tử vào cuối danh sách---*)

  procedure insert_cuoi(var l:tro;X:integer);
    var
      moi,p:tro;
    begin
      new(moi);
      moi^.info:=X;
      if l=nil then begin
                      moi^.link:=l;
                      l:=moi;
                    end
      else
          begin
              p:=l;
              while p^.link<>nil do
                    p:=p^.link;
              moi^.link:=p^.link;
              p^.link:=moi;
          end;
    end;
(*---thủ tục hiện danh sách---*)
  procedure listing_ds(l:tro);
    var
      p:tro;
```

```
    begin
      p:=l;
      while p<>nil do begin
                        write(p^.info:4);
                        p:=p^.link;
                        end;
    end;

(*-chèn phần tử vào vị trí bất kỳ trong danh sách -*)

  procedure insert_giua(var l:tro;k,X:integer);
    var
      moi,p:tro;
      count:integer;
    begin
      new(moi);
      moi^.info:=X;
      if l=nil then begin
                      moi^.link:=l;
                      l:=moi;
                    end
      else
          begin
              p:=l;count:=1;
              while (p^.link<>nil) and (count<k) do
                  begin
                       p:=p^.link;
                       count:=count+1;
                  end;
              moi^.link:=p^.link;
              p^.link:=moi;
          end;
    end;
```

```
(*---chương trình chính---*)

   BEGIN
    repeat
       clrscr;
       writeln;
       writeln(' Bang chon cong viec');writeln;
       writeln('  1. Khoi tao danh sach');
       writeln('  2. Chen vao dau danh sach');
       writeln('  3. Chen vao cuoi danh sach');
       writeln('  4. Chen vao vi tri bat ky');
       writeln('  5. Xem danh sach');
       writeln('  6. Ket thuc');writeln;
       write('   Chon cong viec: ');readln(chon);
       Case chon of
         1:Begin
             khoitao(L);readln;
           end;
         2:Begin
             write('Nhap so X= ');readln(X);
             Insert_dau(L,X);
             writeln('Da chen xong!!!');
             write('Bam Enter de tiep tuc');
             readln
           End;
         3:Begin
             write('Nhap so X= ');readln(X);
             Insert_cuoi(L,X);
             writeln('Da chen xong!!!');
             write('Bam Enter de tiep tuc');
             readln
           End;
         4:Begin
             write('Nhap so X= ');readln(X);
```

```
                write('Cho biet vi tri can chen k= ');
                  readln(k);
                Insert_giua(L,k,X);
                writeln('Da chen xong!!!');
                write('Bam Enter de tiep tuc');
                readln
              End;
            5:Begin
                if l=nil then write('Danh sach rong!!!')
                         else listing_ds(L);
                readln;
              End;
        End;
    until chon=6;
  END.
```

2. Chương trình loại bỏ một nút ra khỏi danh sách nối đơn

Viết chương trình loại bỏ một nút ra khỏi danh sách nối đơn. Với các chức năng:

- Khởi tạo danh sách
- Xoá nút đầu danh sách
- Xoá nút cuối danh sách
- Xoá một nút ở vị trí bất kỳ nhập vào từ bàn phím
- Xem danh sách.

Chương trình được tổ chức dưới dạng menu chức năng.

```
(*-----Khai báo-----*)
uses crt;
   const
    mang:array [1..10] of integer=(5,8,6,7,4,9,1,3,2,23);
   type
        tro=^nut;
```

```
        Nut=record
                info:integer;
                link:tro;
            end;
   var
     L:tro;
     X,k:integer;
     chon:integer;
(*---------------*)
   procedure insert_cuoi(var l:tro;x:integer);forward;
(*---thủ tục khởi tạo danh sách---*)

   procedure khoitao(var l:tro);
     var
        i,X:integer;
        p:tro;
     begin
        l:=nil;
        for i:=1 to 10 do
                  begin
                     x:=mang[i];
                     insert_cuoi(l,x);
                  end;
     end;

(*---chèn them một nút vào cuối danh sách---*)

   procedure insert_cuoi(var l:tro;X:integer);
     var
       moi,p:tro;
     begin
```

```
        new(moi);
        moi^.info:=X;
        if l=nil then begin
                          moi^.link:=l;
                          l:=moi;
                      end
        else
            begin
               p:=l;
               while p^.link<>nil do
                     p:=p^.link;
               moi^.link:=p^.link;
               p^.link:=moi;
            end;
     end;

(*---thủ tục hiện danh sách---*)

   procedure listing_ds(l:tro);
     var
       p:tro;
     begin
       p:=l;
       while p<>nil do begin
                           write(p^.info:4);
                           p:=p^.link;
                       end;
     end;

(*---Xoá nút đầu danh sách---*)

   procedure Delete_dau(var l:tro;var x:integer);
     var
        p:tro;
     Begin
```

```
        if l=nil then writeln('Danh sach rong!!!')
        else begin
                  p:=l;
                  l:=l^.link;
                  x:=p^.info;
                  dispose(p);
                  writeln('Phan tu duoc xoa khoi
                                    danh sach: ',x);
             end
    End;

(*---Xoá nút cuối danh sách---*)

   procedure Delete_cuoi(var l:tro;var x:integer);
     var
        p,q:tro;
     Begin
        if l=nil then writeln('Danh sach rong!!!')
        else begin
                 p:=l;q:=l;
                 if l^.link=nil then
                      begin
                         x:=l^.info;
                         l:=l^.link;
                         dispose(p);
                         writeln('Phan tu duoc xoa
                               khoi danh sach: ',x);
                      end
                 else
                      begin
                         while p^.link<>nil do
                                          p:=p^.link;
                         while q^.link<>p do q:=q^.
                                                link;
                         q^.link:=p^.link;
```

```
                    x:=p^.info;
                    dispose(p);
                    writeln('Phan tu duoc xoa
                        khoi danh sach: ',x);
                end
            end
    End;

(*---Xoá nút ở vị trí k nhập vào từ bàn phím---*)

  procedure Delete_giua(var l:tro;var X:integer);
    var
      p,q:tro;
      k,count:integer;
    begin
      if l=nil then
          writeln('Danh sach rong!!!')
      else
          begin
            write('Cho vi tri can xoa k= ');
              readln(k);
            p:=l;q:=l;count:=1;
            while (p^.link<>nil) and (count<>k) do
                begin
                    p:=p^.link;
                    count:=count+1;
                end;
             if (p^.link=nil) and (count<>k) then
                writeln('Khong tim thay nut o vi
                                        tri k')
             else begin
                    if p<>l then
                        while q^.link<>p do
```

```
                                        q:=q^.link
                    else l:=l^.link;
                         q^.link:=p^.link;
                         x:=p^.info; dispose(p);
                         writeln('Phan tu duoc xoa
                             khoi danh sach: ',x);
                end
        end;
    end;
(*-------------*)
  BEGIN
   repeat
      clrscr;
      writeln;
      writeln('          Bang chon cong viec');
      writeln;
      writeln('       1. Khoi tao danh sach');
      writeln('       2. Xoa nut dau danh sach');
      writeln('       3. Xoa nut cuoi danh sach');
      writeln('       4. Xoa nut o vi tri bat ky');
      writeln('       5. Xem danh sach');
      writeln('       6. Ket thuc');writeln;
      write('     Chon cong viec: ');readln(chon);
      Case chon of
        1:Begin
            khoitao(L);
            writeln('Danh sach da duoc khoi tao!!!');
            writeln('Bam Enter de tiep tuc');
            readln;
          end;
        2:Begin
            Delete_dau(l,x);
            write('Bam Enter de tiep tuc');
```

```
            readln
          End;
        3:Begin
            Delete_cuoi(l,x);
            write('Bam Enter de tiep tuc');
            readln
          End;
        4:Begin
            Delete_giua(l,x);
            write('Bam Enter de tiep tuc');
            readln
          End;
        5:Begin
            if l=nil then
                 writeln('Danh sach rong!!!')
            else listing_ds(L);
            readln;
          End;
    End;
 until chon=6;
END.
```

3. Chương trình bổ sung và loại bỏ một nút của danh sách nối kép

Gồm các chức năng:

- Khởi tạo danh sách
- Bổ sung một nút mới vào trước nút thứ k
- Loại bỏ nút thứ k khỏi danh sách
- Hiển thị danh sách

Chương trình được tổ chức dưới dạng menu chức năng.

```
uses crt;
type
      tro=^nut;
      Nut=record
```

```
                info:integer;
                Lptr,Rptr:tro;
              end;
  var
    L,R:tro;
    X,k:integer;
    chon:integer;

(*--------------*)

  procedure khoitao(var l,r:tro);
    begin
       L:=nil;R:=nil;
       writeln('Danh sach da duoc khoi tao!!!');
       writeln('Bam phim Enter de tiep tuc!!!');
    end;

(*--------------*)
{chen mot nut vao truoc nut tro boi nut cuc trai L}

  procedure insert_dau(var l,r:tro;X:integer);
    var
      moi:tro;
    begin
      new(moi);
      moi^.info:=X;
      if l=nil then begin
                       moi^.lptr:=nil;
                       moi^.Rptr:=nil;
                       l:=moi;r:=moi;
                    end
      else
          begin
              moi^.rptr:=l;
```

```
                moi^.lptr:=l^.lptr;
                l^.lptr:=moi;
                l:=moi;
            end;
    end;

(*---------------*)
{chen mot nut vao sau nut tro boi nut cuc phai R}

   procedure insert_cuoi(var l,r:tro;X:integer);
     var
       moi:tro;
     begin
       new(moi);
       moi^.info:=X;
       if l=nil then begin
                       moi^.lptr:=nil;
                       moi^.Rptr:=nil;
                       l:=moi;r:=moi;
                     end
       else
            begin
                moi^.rptr:=r^.rptr;
                moi^.lptr:=r;
                r^.rptr:=moi;
                r:=moi;
            end;
     end;

(*--------------*)

   procedure listing_ds(l,r:tro);
     var
       p:tro;
     begin
```

```
            p:=l;
            writeln;
            if l=nil then begin
                              writeln('Danh sach rong!');
                              writeln('Bam Enter de ve
                                                    menu');
                              readln; exit;
                            end;
             writeln('Duyet danh sach tu trai -> phai');
             while p<>nil do begin
                              write(p^.info:4);
                              p:=p^.rptr;
                             end;
            writeln;
            writeln('Duyet danh sach tu phai -> trai');
            p:=r;
            while p<>nil do begin
                               write(p^.info:4);
                               p:=p^.lptr;
                            end;
            readln;
          end;

   (*-------------*)
  {chen mot nut vao truoc nut o vi tri thu k trong danh
sach}
      procedure insert_giua(var l,r:tro;k,X:integer);
        var
          moi,p:tro;
          count:integer;
        begin
          new(moi);
          moi^.info:=X;
          if l=nil then begin
                          moi^.lptr:=nil;
                          moi^.Rptr:=nil;
```

```
                        l:=moi;r:=moi;
                    end
        else
            begin
                p:=l;count:=1;
                while (p<>r) and (count<k) do
                    begin
                        p:=p^.rptr;
                        count:=count+1;
                    end;
                if p=L then insert_dau(l,r,x)
                    else
                        begin
                            moi^.Rptr:=p;
                            moi^.Lptr:=p^.lptr;
                            p^.lptr:=moi;
                            moi^.Lptr^.Rptr:=moi;
                        end
            end;
    end;

(*-------------*)
{Xoa nut cuc trai duoc tro boi L}

  procedure Delete_dau(var l,r:tro;var x:integer);
    var
      p:tro;
    Begin
      If l=nil then
        begin
          writeln('Danh sach rong!!!');
          writeln('Bam Enter tro ve menu');
          readln;
        end
      else
```

```
            begin
                p:=L;
                L:=L^.rptr;
                L^.Lptr:=nil;
                X:=p^.info;
                dispose(p);
                writeln('Phan tu duoc xoa khoi danh
                                        sach X= ',X);
                writeln('Da xoa xong! Bam Enter de ve
                                                menu');
                readln;
            end;
        if l=nil then r:=nil;
    End;

(*-------------*)
{Xoa nut cuc phai duoc tro boi R}
   procedure Delete_cuoi(var l,r:tro;var x:integer);
      var
         p:tro;
      Begin
         If R=nil then
             begin
                 writeln('danh sach rong!!!');
                 writeln('Bam Enter tro ve menu');
                 readln;
             end
         else
             begin
               p:=R;
               R:=R^.lptr;
               R^.Rptr:=nil;
               X:=p^.info;
               dispose(p);
```

```
                writeln('Phan tu duoc xoa khoi danh
                                          sach X= ',X);
                writeln('Da xoa xong! Bam Enter de ve
                                                menu');
                readln;
             end;
          if R=nil then L:=nil;
       End;

(*--------------*)
{Xoa nut tai vi tri thu k trong danh sach}
   procedure Delete_k(var l,r:tro; k:integer;var
                                          x:integer);
      var
         p:tro;
         count:integer;
      Begin
                p:=l;count:=1;
                while (p<>r) and (count<k) do
                     begin
                          p:=p^.rptr;
                          count:=count+1;
                     end;
                if count<>k then
                     begin
                         writeln('Khong tim thay nut
                                              thu ',k);
                         writeln('Bam Enter de ve
                                                menu');
                         readln; exit;
                     end;
                if p=l then begin
                                delete_dau(l,r,x);
                                exit;
                            end;
                if p=r then begin
                                delete_cuoi(l,r,x);
```

```
                        exit;
                      end;
                x:=p^.info;
                p^.lptr^.rptr:=p^.rptr;
                p^.rptr^.lptr:=p^.lptr;
                dispose(p);
                writeln('Phan tu duoc xoa khoi danh
                                        sach X= ',X);
                writeln('Da xoa xong! Bam Enter de ve
                                              menu');
                readln;
    End;

(*---------------*)

  BEGIN
   repeat
      clrscr;
      writeln;
      writeln(' Bang chon cong viec');writeln;
      writeln('      1. Khoi tao danh sach');
      writeln('      2. Chen vao dau danh sach');
      writeln('      3. Chen vao cuoi danh sach');
      writeln('      4. Chen vao vi tri bat ky');
      writeln('      5. Xoa nut dau danh sach');
      writeln('      6. Xoa nut cuoi danh sach');
      writeln('      7. Xoa nut o vi tri bat ky');
      writeln('      8. Xem danh sach');
      writeln('      9. Ket thuc');writeln;
      write('        Chon cong viec: ');readln(chon);
      Case chon of
        1:Begin
            khoitao(L,R);readln;
          end;
        2:Begin
```

```
      write('Nhap so X= ');readln(X);
      Insert_dau(L,R,X);
      writeln('Da chen xong!!!');
      write('Bam Enter de tiep tuc');
      readln
    End;
3:Begin
      write('Nhap so X= ');readln(X);
      Insert_cuoi(L,R,X);
      writeln('Da chen xong!!!');
      write('Bam Enter de tiep tuc');
      readln
    End;
4:Begin
      write('Nhap so X= ');readln(X);
      write('Cho biet vi tri can chen k=
                                  ');readln(k);
      Insert_giua(L,R,k,X);
      writeln('Da chen xong!!!');
      write('Bam Enter de tiep tuc');
      readln
    End;
5:Begin
      Delete_dau(l,r,x);
    End;
6:Begin
      Delete_cuoi(l,r,x);
    End;
7:Begin
      if r=nil then
         begin
              writeln('Danh sach rong!!!');
              writeln('Bam Enter de tro ve
                                      menu');
              readln;
         end
      else
```

```
                begin
                    write('Vi tri can xoa k=
                                        ');readln(k);
                    Delete_k(l,r,k,x);
                end
            End;
        8:Begin
            listing_ds(L,R);
            End;
    End;
 until chon=9;
END.
```

Bài tập

1. Lập giải thuật thực hiện các phép sau đây đối với danh sách nối đơn mà nút đầu tiên của nó được trỏ bởi L.

a) Tính số lượng các nút của danh sách.

b) Tìm tới nút thứ k trong danh sách, nếu có nút thứ k thì in ra trường info của nút đó, nếu không có thì thông bào là không có nút thứ k.

c) Bổ sung một nút vào sau nút thứ k

d) Loại bỏ nút đứng trước nút thứ k.

e) Cho thêm con trỏ M trỏ tới một nút có trong danh sách nói trên và một danh sách nối đơn khác có nút đầu tiên được trỏ bởi P. Hãy chèn danh sách P này vào sau nút trỏ bởi M.

f) Tách thành hai danh sách mà danh sách sau trỏ bởi M (cho như ở câu e).

g) Đảo ngược danh sách đã cho (tạo một danh sách trỏ bởi Q mà các nút móc nối theo thứ tự ngược lại so với L).

2. Cho một danh sách nối đơn có nút đầu danh sách trỏ bởi P. Giá trị của trường Info trong các nút giả sử là các số khác nhau và các nút đã được sắp xếp theo thứ tự tăng dần của giá trị này. Hãy lập giải thuật:

a) Bổ sung một nút mới, mà trường Info có giá trị bằng X vào danh sách.

b) Loại bỏ nút mà trường Info có giá trị bằng K cho trước.

3. Cho L và R là hai con trỏ trỏ tới nút cực trái và nút cực phải của một danh sách nối kép. Hãy lập giải thuật bổ sung và loại bỏ để danh sách đó hoạt động như một Queue.

4. Cho một danh sách nối đơn có nút đầu danh sách trỏ bởi L. Hãy lập giải thuật bổ sung và loại bỏ để danh sách đó hoạt động như một Stack.

Chương 5

CÂY

Mục tiêu:

- Người học hiểu được thế nào là cây; cây con; cây rỗng và các loại nút của cây như nút gốc; nút lá; ...
- Người học hiểu được một số khái niệm khác quan trọng về cây như bậc của nút; bậc của cây; nút trung gian; nút kết thúc; mức của nút; chiều cao của cây; nút trước; nút sau; nút cha; nút con và đường đi của cây.
- Người học hiểu được thế nào là cây nhị phân và tính chất của nó. Từ đó người học biết các cách biểu diễn cây nhị phân.
- Người học biết được các phép duyệt cây khác nhau như phép duyệt cây theo thứ tự trước; duyệt theo thứ tự giữa và duyệt theo thứ tự sau.
- Người học được cung cấp một số thủ tục về đệ quy và không đệ quy mẫu mô tả các giải thuật duyệt cây theo thứ tự trước; theo thứ tự giữa và theo thứ tự sau để củng cố kiến thức về cây và giúp người học hiểu rõ về cây và các phép duyệt cây.

I. ĐỊNH NGHĨA VÀ CÁC KHÁI NIỆM

1. Định nghĩa

Cây là một tập hữu hạn các nút (mục tin) trong đó

- Có một nút đặc biệt gọi là gốc.
- Các nút còn lại được phân chia thành m nhóm ($m \geq 0$), mỗi nhóm lại tương ứng với một cấu trúc cây và được gọi là cây con.
- Giữa các nút có quan hệ phân cấp gọi là "quan hệ cha con".

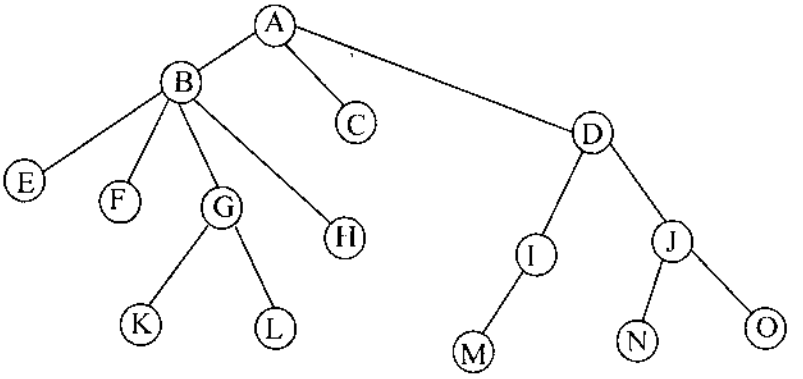
* *Định nghĩa theo đệ quy*

1. Một nút là một cây. Nút đó cũng là gốc của cây ấy.

2. Nếu n là một nút và T_1, T_2, ... , T_K là các cây với n_1, n_2..., n_K lần lượt là các gốc thì cây T mới được tạo lập bằng cách cho nút n trở thành cha của các nút n_1, n_2,..., n_K. Lúc đó n_1, n_2, ... , n_K là con của nút n.

Để thuận tiện, người ta quy ước cho phép tồn tại một cây không có nút nào một cây như vậy được gọi là cây rỗng.

Biểu diễn một cây có dạng như sau:



Hình 5.1

2. Một số khái niệm

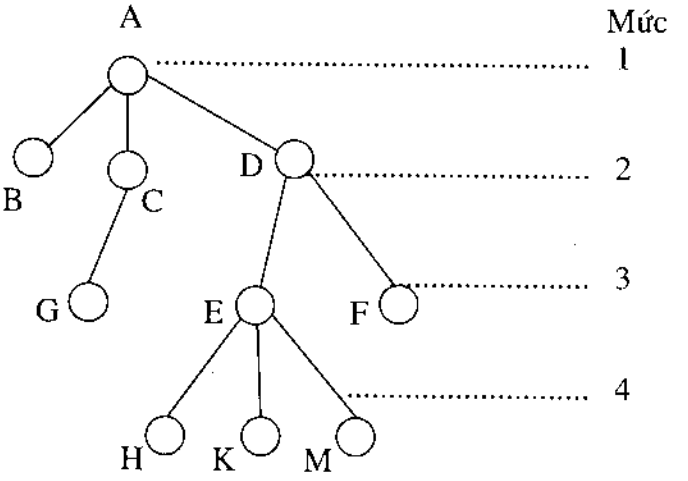
* Số các con của một nút gọi là *cấp* của nút đó. Nút có cấp bằng không gọi là nút *lá* hay *nút tận cùng*.

Cấp cao nhất của nút trên cây gọi là *cấp* của *cây* đó.

* Gốc của cây có số *mức* là 1. Nếu nút cha có số mức là i thì nút con có số mức là i + 1.

* *Chiều cao* hay *chiều sâu* của một cây là số mức lớn nhất của nút có trên cây đó.

Ví dụ:



Hình 5.2

Cấp của nút A là 3, của nút B là 0, nút D là 2.

Cấp của cây là 3

Nếu n_1, n_2,...,n_k là dãy các nút mà n_i là cha của n_{i+1} với $1 \le i < k$, thì dãy đó gọi là *đường đi* từ n_1 tới n_k. *Độ dài của đường đi* bằng số nút trên đường đi trừ đi 1.

*. Nếu thứ tự các cây con của một nút được coi trọng thì cây đó được gọi là *cây có thứ tự*, ngược lại là *cây không có thứ tự*. Thường thứ tự các cây con của một nút được đặt từ trái qua phải.

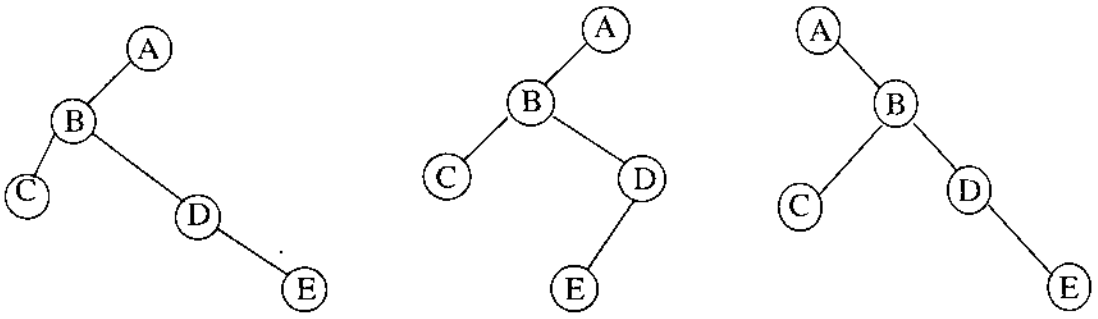
* Nếu có một tập hữu hạn các cây phân biệt thì ta gọi tập đó là *rừng*. Khái niệm về rừng ở đây ta hiểu một cách tương đối vì nếu ta bỏ nút gốc của một cây thì ta sẽ có một rừng.

II. CÂY NHỊ PHÂN

1. Định nghĩa và tính chất

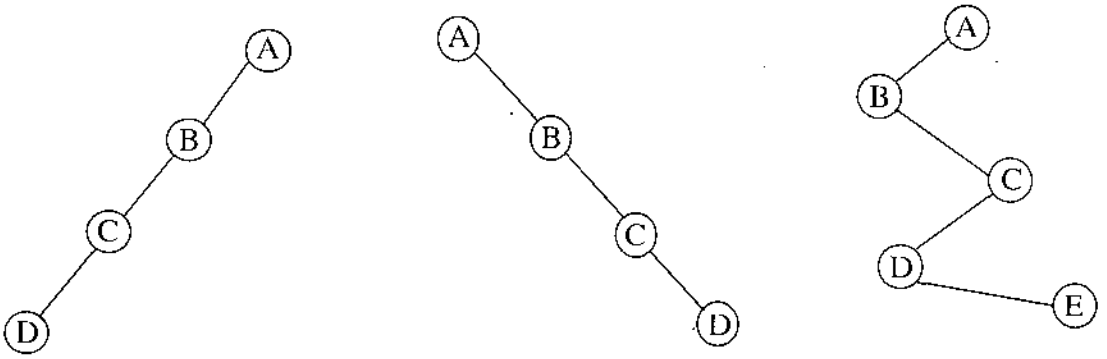
Cây nhị phân là một dạng quan trọng của cấu trúc cây. Nó có đặc điểm là: mọi nút trên cây chỉ có tối đa là 2 con. Đối với cây con của một nút người ta cũng phân biệt *cây con trái* và *cây con phải*. Như vậy cây nhị phân là cây có thứ tự.

Ví dụ:



Hình 5.3

* Một số dạng đặc biệt của cây:



(a) *(b)* *(c)*

Hình 5.4

Các hình trên là cây suy biến, thực chất nó có dạng của một danh sách tuyến tính.

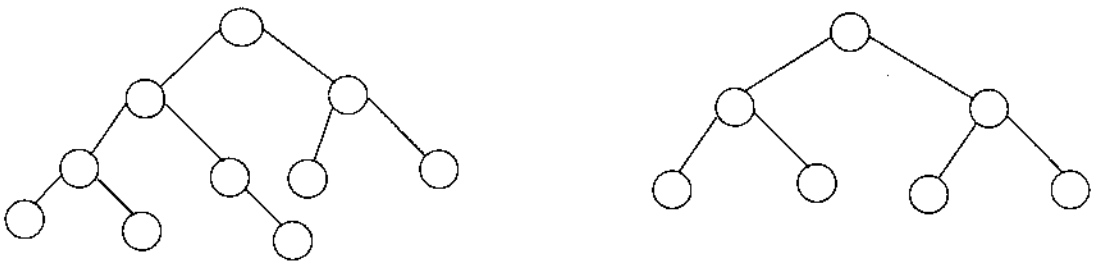
a) Được gọi là cây lệch trái

b) Được gọi là cây lệch phải

c) Được gọi là cây zic-zac

Một cây nhị phân được gọi là *hoàn chỉnh* nếu các nút ứng với các mức (trừ mức cuối) đều có hai con.

Một cây nhị phân được gọi là *đầy đủ* nếu các nút ứng với các mức đều có hai con.



(a) *(b)*

Hình 5.5

a) Cây hoàn chỉnh; b) Cây đầy đủ

Nhận xét: Các cây nhị phân có số nút như nhau thì cây suy biến có chiều cao lớn nhất. Cây nhị phân đầy đủ có chiều cao nhỏ nhất.

* *Một số tính chất*

1) Số lượng tối đa các nút ở mức i trên một cây nhị phân là 2^{i-1} ($i \geq 1$).

2) Cây nhị phân hoàn chỉnh có *n* nút có chiều cao *h* là:

$$2^h - 1 \ (h \geq 1)$$

$$h = \lceil \log_2 (n+1) \rceil$$

Quy ước:

$\lceil X \rceil$: số nguyên trên của X

$\lfloor X \rfloor$: số nguyên dưới của X

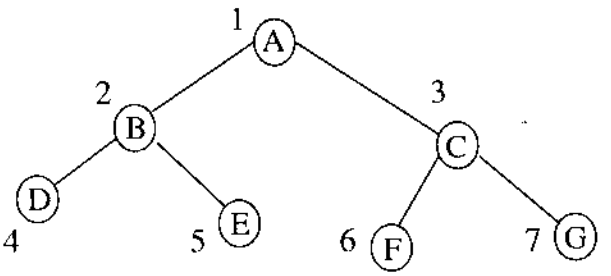
$\lfloor X \rfloor \leq X \leq \lceil X \rceil$

2. Biểu diễn cây nhị phân

2.1. Lưu trữ kế tiếp

Nếu có một cây hoàn chỉnh đầy đủ, ta có thể đánh số các nút trên cây theo thứ tự lần lượt từ mức 1 trở lên. Hết mức này đến mức khác từ trái qua phải.

Ví dụ:



Hình 5.6

- Con nút thứ i là 2i và 2i+1
- Cha nút thứ j là j div 2

Như vậy ta sẽ lưu trữ cây dạng 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 với cây nhị phân; với cách lưu trữ này nếu biết được địa chỉ của nút cha sẽ tính được địa chỉ của nút con và ngược lại.

Như vậy đối 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]

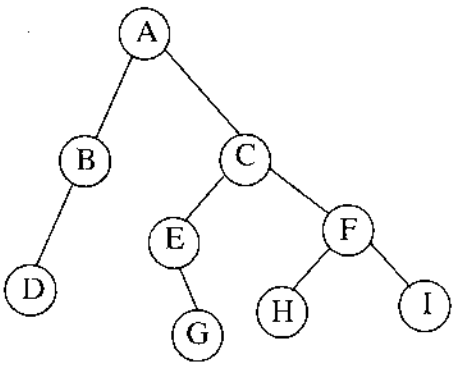
Ngoài ra cây luôn biến động nghĩa là phép bổ sung và loại bỏ thường xuyên tác động —> Cách lưu trữ này có nhiều nhược điểm. Nếu cây không đầy đủ thì lưu trữ sẽ bị lãng phí.

2.2. Lưu trữ móc nối

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

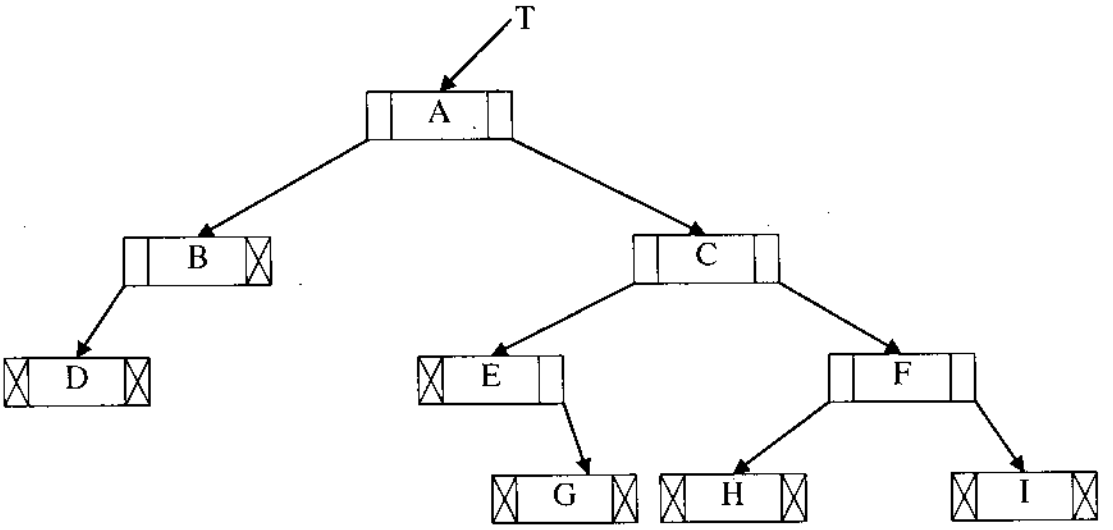
LTRO	DATA	RTRO

Trường DATA ứng với thông tin của nút.
Trường LTRO ứng với con trỏ, trỏ tới cây con trái.
Trường RTRO ứng với con trỏ, trỏ tới cây con phải.
Ví dụ:
Cây nhị phân sau:



Hình 5.7

Sẽ có dạng lưu trữ như dưới đây:



Hình 5.8

Để có thể truy nhập các nút ở trên cây cần con trỏ T trỏ tới nút gốc của cây đó.

Quy ước: Cây nhị phân rỗng thì T = null;

III. PHÉP DUYỆT CÂY NHỊ PHÂN

Thông thường ta hay phải thực hiện các phép xử lý trên mỗi nút của cây theo một thứ tự nào đó.

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

Ta có 3 phép duyệt cây khác nhau đối với cây nhị phân.

Các phép đó được định nghĩa một cách đệ quy như sau:

1. Duyệt theo thứ tự trước

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

2. Duyệt cây theo thứ tự giữa

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

3. Duyệt cây theo thứ tự sau

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

Ví dụ:

Xét cây nhị phân ở trên, dãy các tên ứng với nút được thăm trong các phép duyệt sẽ là:

a) Theo thứ tự trước

A B D C E G F H I

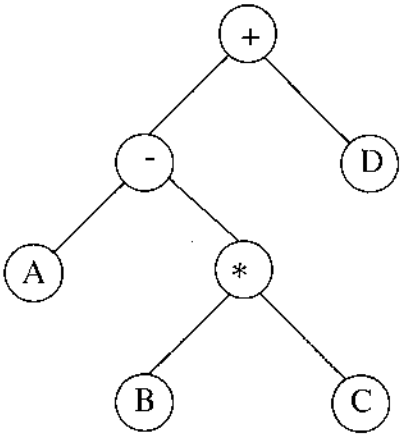
b) Theo thứ tự giữa

D B A E G C H F I

c) Theo thứ tự sau

D B G E H I F C A

Một ví dụ khác xét cây



Hình 5.9

Duyệt trước ta được + - A * B C D

Duyệt giữa ta được A - B * C + D

Duyệt sau ta được A B C * - D +

Như vậy các phép duyệt nêu trên đối với cây nhị phân biểu diễn biểu thức số học sẽ cho ta các dạng ký pháp Ba Lan của biểu thức số học.

Nếu viết dưới dạng thủ tục đệ quy thì các giải thuật của các phép duyệt cây nhị phân sẽ như sau:

Procedure DUYETTRUOC(T);

{T là con trỏ trỏ tới gốc cây, phép thăm ở đây được đặc trưng bởi in giá trị trường DATA của nút}

1. **If** T <> null **then**

```
Begin
   Write(DATA(T));
   Call DUYETTRUOC(Ltro(T));
   Call DUYETTRUOC (Rtro(T));
End;
```

2. **Return**.

Procedure DUYETSAU(T);

1. **if** T <> null **then**

```
Begin
   Call DUYETSAU(Ltro(T));
   Call DUYETSAU(Rtro(T));
   Write(DATA(T));
End;
```

2. **Return**

Tương tự, ta có thể viết thủ tục DUYETGIUA.

Procedure DUYETGIUA(T);

1. **if** T <> null **then**

```
Begin
   Call DUYETGIUA(Ltro(T));
   Write(DATA(T));
   Call DUYETGIUA(Rtro(T));
End;
```

2. **Return**

Ta thấy viết các thủ tục đệ quy này là khá đơn giản vì nó phản ánh đúng dạng các định nghĩa đã nêu của các phép duyệt bằng các câu lệnh tương ứng.

Nếu muốn mô tả giải thuật dưới dạng không đệ quy thì có thể sử dụng Stack, để nạp vào đó các địa chỉ các nút cần thiết khi "đi xuống" theo một nhánh nào đó và lấy các địa chỉ ra khỏi Stack, để xác định đường "đi lên" trong khi thực hiện phép duyệt.

Sau đây là giải thuật không dùng đệ quy

Procedure DUYETTRUOC(T);

{T là con trỏ trỏ tới gốc, S là một Stack (tổ chức theo kiểu kế tiếp), TOP là con trỏ trỏ tới đỉnh Stack. Biến trỏ P trỏ tới nút đang được xem xét; trong phép duyệt này khi *thăm* gốc xong ta đưa *địa chỉ các nút con phải* vào Stack để sau này có thể quay trở lại các nút này trước khi đi xuống *cây con trái*}

```
1. {Khởi đầu}
 If T=Null then
   Begin
        Write('Cây rỗng');
        Return;
   End
Else
   Begin
        Top:=0;
        Call BOXUNG(S,TOP,T);
   End;
2. {Thực hiện duyệt}
While Top>0 do
   Begin
        P:=Loaibo(S,TOP); {Lấy một phần tử của Stack ra}
3. {Thăm nút rồi xuống cây con trái}
While  P<> Nil do
   Begin
        Write(DATA(P)); {Thăm P}
        If  Rtro(P) <> Null then
              Call Boxung(S,Top,Rtro(P));
        {Lưu trữ địa chỉ gốc cây con phải}
              P:=Ltro(P); {Xuống con trái}
   End;
   End;
4. Return.
```

Ta cũng có thể diễn đạt lại giải thuật duyệt trước như sau:

```
Procedure DUYETRUOC(p)
begin
  sp:=0;
  while p<> null do
    begin
      write(DATA(p))
        if Rtro(p) <> null then
          begin
            sp :=sp +1;
            stack[sp] := Rtro(p)
          end;
    end;
if  Ltro(p) <> null then p:=Ltro(p)
    else {lấy từ stack ra}
          if sp = 0 then giải thuật kết thúc
                else begin
                        p:= stack[sp]; sp := sp-1;
                     end;
end;
```

Giải thuật kết thúc khi Stack rỗng.

Ta minh hoạ giải thuật này như sau: giả sử cây được chứa trong bộ nhớ như dưới đây:

Địa chỉ	Nội dung	Ltro	Rtro
1	A	2	3
2	B	4	5
3	H	8	9
4	C	N	N
5	D	6	7
6	E	N	N
7	F	N	N
8	K	N	N
9	L	N	N

Diễn biến giải thuật như sau:

Lần	Nội dung in	Stack	p	Ltro	Rtro
1	A	3	1	2	3
2	AB	35	2	4	5
3	ABC	35	4	null	null
4	ABCD	37	5	6	7
5	ABCDE	37	6	null	null
6	ABCDEF	3	7	null	null
7	ABCDEFH	9	3	8	9
8	ABCDEFHK	9	8	null	null
9	ABCDEFHKL	null	9	null	null

Procedure DUYETGIUA(T);

{Ta dùng Stack để chứa đường đã đi qua để sau này quay trở lại tiếp tục phép duyệt}

```
sp: = 0; p: = GOC;
repeat
   while p <> null do
         begin
            sp: = sp +1; stack[sp]: = p;
            p: = Ltro(p);
         end;
  if sp <> 0 then
    begin
          p: = stack[sp]; sp: = sp -1;
          write(DATA(p));
          p: = Rtro(p);
          ok: = true;
    end
  else ok: = false;
Until not ok;
```

Diễn biến giải thuật như sau:

p cũ	p mới	Stack	Nội dung in
A	B	A	
B	C	AB	
C	null	ABC	
C	null	AB	C
B	D	AD	CB
D	E	AD	CB
E	null	ADE	CB
E	null	AD	CBE
D	F	A	CBED
F	null	AF	CBED
F	null	A	CBEDF
A	H	null	CBEDFA
H	K	H	CBEDFA
K	null	HK	CBEDFA
K	null	H	CBEDFAK
H	L	null	CBEDFAKH
L	null	L	CBEDFAKH
L	null	null	CBEDFAKHL
null			

Duyệt sau khó hơn cả; ở đây ta đưa cả nút gốc và nút phải vào chồng, nút phải đưa vào sau (để lấy ra trước) và để phân biệt giữa nút gốc và nút phải ta cho nút gốc chứa trong Stack có giá trị âm.

Procedure DUYETSAU(T);

{Trong phép duyệt này nút gốc chỉ được thăm sau khi đã duyệt xong hai cây con của nó; như vậy chỉ khi từ cây con phải lên gặp gốc mới được thăm, chứ

không phải ở lần gặp khi từ con trái lên; do đó để phân biệt ta đưa thêm dấu âm vào địa chỉ để đánh dấu cho lần đi lên từ con trỏ phải}

```
1. {Khởi đầu}
   If T=Null then
     Begin
          Write('Cây rỗng');
          Return;
     End
  Else Begin
          Top:= 0;
          P:= T;
        End;
2. {Thực hiện duyệt}
While True do {lưu trữ địa chỉ gốc và xuống con trái}
   Begin
          While P <> Null do
                Begin {Lưu trữ gốc và xuống cây con trái}
                      Call Boxung(S,TOP,P);
                      P:= Ltro(P);
                End;
3. {Thăm nút mà con trái, con phải đã duyệt}
          While S[Top] < 0 do
                Begin
                      P:= Loaibo(S,TOP);
                      Write(DATA(P));
                      If Top = 0 then Return
                End;
4. {Xuống cây con phải và đánh dấu}
          P:= Rtro(S[Top]);
          S[Top]:= - S[Top]; {Dấu âm cho lần đi lên từ con phải}
   End;
5. Return.
```

Ta có thể diễn đạt giải thuật một cách khác như sau:

```
Sp:= 0;
p:= GOC; ps:= 1;
stack[0]:= null;
repeat
   while (p<>null) and (ps >0) do
         begin
               sp:= sp +1;
               stack[sp]:= Rtro(p);
               stsign[sp]:= 1;
         end;
   p:= Ltro(p)
   if p <> null then write(DATA(p));
   p := stack[sp]; ps:= stsign[sp];
   sp:= sp-1;
until  p= null;
```

IV. MỘT SỐ CHƯƠNG TRÌNH MINH HOẠ

* Chương trình dựng cây nhị phân móc nối và các phép duyệt cây

```
Uses crt,dos;
Type
       tro = ^nut;
       nut = record
                    gtri : integer;
                    t_trai,t_phai :tro
              end;
Var
      i,j,n: integer;
      cay:tro;
      mn1: char;
      nu,sn:integer;
      mn: array [1..25] of integer;
      ktmn1:boolean;

(*---------------*)
```

```
   Procedure khoi_tao(var t: tro);
      begin
            T:=nil;
      end;

(*--------------*)

   Procedure DUYET_TRUOC(t:tro);
      begin
          if t<>nil then
             begin
                  write(t^.gtri,' ');
                  duyet_truoc(t^.t_trai);
                  duyet_truoc(t^.t_phai);
             end;
      end;

(*--------------*)

   Procedure DUYET_GIUA(t:tro);
      begin
         if t<>nil then
            begin
                 duyet_giua(t^.t_trai);
                 write(t^.gtri,' ');
                 duyet_giua(t^.t_phai);
            End;
      end;

(*--------------*)

   Procedure DUYET_SAU(t:tro);
      begin
         if t<>nil then
```

```
            begin
                  duyet_sau(t^.t_trai);
                  duyet_sau(t^.t_phai);
                  write(t^.gtri,' ');
            End;
      end;

(*--------------*)

   Procedure Dung_cay_np(var t:tro; i:integer);
      Var
          x,i1,i2:integer;
      Begin
          x:= mn [i];
          new(t);
          t^.gtri:= x;
          i1:= 2*i;
          i2:= 2*i+1;
          if i1<= sn then
              begin
                  dung_cay_np(t^.t_trai,i1);
              end
          else
              t^.t_trai:= nil;
          if i2 <= sn then
              begin
                  dung_cay_np(t^.t_phai,i2);
              end
          else
              t^.t_phai:= nil;
      End;

(*--------------*)

   Procedure Nhan_tp(t:tro);
```

```
    Var
        p: tro;
        dem: integer;
    begin
        p:=t;
        if p<>nil then
            begin
                duyet_giua(t^.t_trai);
                if dem = i*2 then
                    begin
                        write(p^.gtri,' ');
                        dem := dem+1;
                        i := i*2;
                    end;
                 duyet_giua(t^.t_phai);
            end;
    end;

(*---------------*)

  BEGIN
    Repeat
      clrscr;ktmn1:=false;
      textcolor(yellow) ;
      gotoXY(30,3); write(' CAY NHI PHAN');
      textcolor(lightcyan) ;
      gotoXY(25,4); write('          ----- ');
      gotoXY(22,6);
       write('1. Dung cay duoi dang moc noi');
      gotoXY(22,7);
       write('2. Duyet cay theo thu tu truoc');
      gotoXY(22,8);
       write('3. Duyet cay theo thu tu giua ');
      gotoXY(22,9);
       write('4. Duyet cay theo thu tu sau  ');
```

```
gotoXY(22,10);
 write('5. In nhan cua nut con trai    ');
gotoXY(22,11);write('6. Ket thuc');
textcolor(red) ;
gotoXY(22,13); write('Chon viec so: ');
textcolor(lightcyan) ;
readln(mn1);
case mn1 of
     '1' : Begin
           clrscr;
           n:=1;
           khoi_tao(cay);
           writeln(' ');
           textcolor(yellow) ;
           gotoXY(22,5);
           write('Cho so phan tu <=25 phan tu:');
           readln(sn); clrscr;
           textcolor(yellow) ;
           gotoXY(25,5);
           writeln('Dua vao ',sn,' so nguyen: ');
           gotoXY(30,6);writeln(' ------ ');
           writeln;
           textcolor(lightcyan) ;
           for i:=1 to sn do
                begin
                  write('     Pt - ',i, ' : ');
                  read(nu);
                  MN[i]:=nu;
                end;
             readln;
             sn:=i;
            i:=1;
             dung_cay_np(Cay,i);
             readln;
         End;
```

```
'2' : Begin
       clrscr;
       textcolor(yellow) ;
       gotoXY(5,5);
       writeln('Duyet cay theo thu tu truoc:');
       writeln;
       textcolor(lightcyan) ;
       DUYET_TRUOC(cay);
       readln;
      End;
'3' : Begin
       clrscr;
       gotoXY(5,5);
       textcolor(yellow) ;
       write('duyet cay theo thu tu giua:');
       textcolor(lightcyan) ;
       writeln;
       DUYET_GIUA(cay);
       readln;
      End;
'4' : Begin
       clrscr;
       gotoXY(5,5);
       textcolor(yellow) ;
       write('duyet cay theo thu tu sau:');
       textcolor(lightcyan) ;
       writeln;
       duyet_sau(cay);
       readln;
      End;
'5' : Begin
       i:=1;
       clrscr;
       textcolor(yellow) ;
       gotoXY(10,5);
       writeln('Nhan trai,phai cua cay nhi
                                    phan:');
```

```
                    gotoXY(10,6);
                    writeln('   ------   ');
                    textcolor(lightcyan) ;
                    writeln;
                    write('              ');
                    Nhan_tp(cay);
                    readln;
                    End;
              '6' : ktmnl := true
        End;
      Until ktmnl= true;
End.
```

Bài tập

1. Vẽ cây nhị phân biểu diễn các biểu thức sau đây và viết chúng dưới dạng tiền tố, hậu tố.

a) (a * b + c)/(d - e * f)

b) A/(B+C) + D * E - A * C

c) $(x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2) \vee x_3$

Với $\wedge$ là phép và (and)

$\vee$ là phép hoặc (or)

$\neg$ là phép không (not)

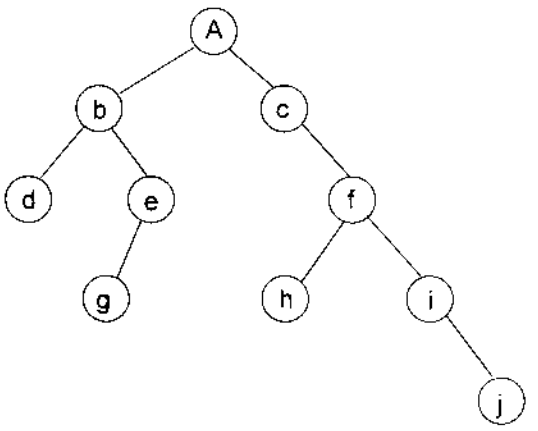
2. Cho cây nhị phân

Hãy viết các nút được thăm khi duyệt cây này

a) Theo thứ tự trước

b) Theo thứ tự giữa

c) Theo thứ tự sau



3. Chứng tỏ rằng nếu cho biết dãy các nút được thăm của một cây nhị phân khi duyệt theo thứ tự trước và thứ tự giữa thì có thể dựng được cây nhị phân đó.

Điều này còn đúng nữa không đối với thứ tự trước và thứ tự sau? Đối với thứ tự giữa và thứ tự sau?

Chương 6

ĐỒ THỊ VÀ MỘT VÀI CẤU TRÚC PHI TUYẾN

Mục tiêu:

- Người học hiểu được thế nào là đồ thị; thế nào là đỉnh và cung của đồ thị từ đó có thể lấy được các ví dụ thực tế về đồ thị.

- Người học biết được một số khái niệm liên quan đến đồ thị như đồ thị định hướng; đồ thị không định hướng; định hướng liên thông và định hướng không liên thông.

- Người học biết các cách biểu diễn đồ thị như biểu diễn bằng đồ thị lân cận và danh sách lân cận.

- Người học biết được các phép duyệt một đồ thị với các giải thuật minh hoạ. Đó là các phép tìm kiếm theo chiều rộng và tìm kiếm theo chiều sâu với các đồ thị.

- Người học hiểu được khái niệm về cây khung và cây khung với giá trị cực tiểu. Mặt khác người học còn được cung cấp một số giải thuật minh hoạ về cây khung.

- Chương này còn cung cấp một số bài tập mẫu cùng lời giải để cho người học củng cố kiến thức về đồ thị, về cây khung.

I. ĐỊNH NGHĨA VÀ CÁC KHÁI NIỆM VỀ ĐỒ THỊ

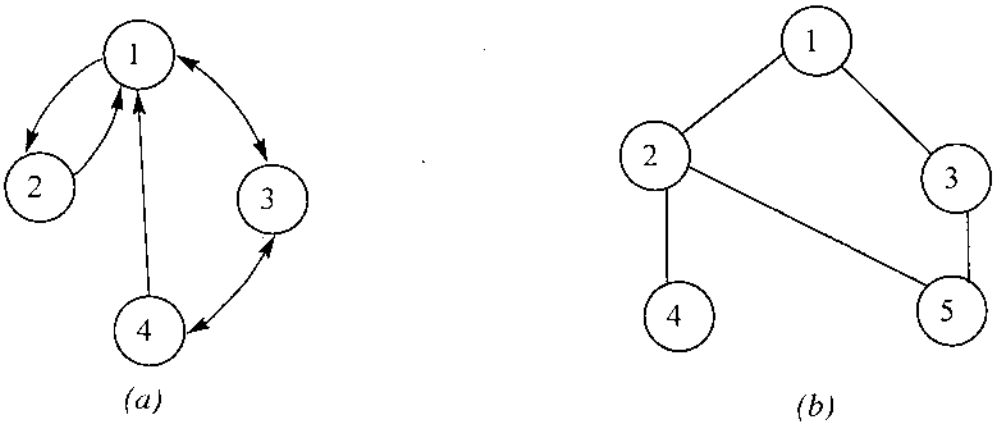
Một đồ thị G (V, E) bao gồm một tập hữu hạn V các nút và một tập hữu hạn E các cặp đỉnh gọi là cung. Sau đây là một số khái niệm:

- Nếu (v_1, v_2) là một cặp đỉnh thuộc E thì ta nói có một cung nối v_1 và v_2.

- Nếu cung (v_1, v_2) khác cung (v_2, v_1) thì ta nói có một đồ thị định hướng. Lúc đó (v_1, v_2) gọi là cung định hướng từ v_1 tới v_2.

- Nếu thứ tự các nút trên cây không được coi trọng thì ta có đồ thị không định hướng.

Thí dụ ta có thể biểu diễn đồ thị bằng hình vẽ như sau:



Hình 6.1

a) Đồ thị có hướng; b) Đồ thị không định hướng

Mạch điện, mạng điện, mạng máy tính,... là các ví dụ thực tế của đồ thị. Cây là một trường hợp đặc biệt của đồ thị.

Nếu (v_1, v_2) là một cung trong tập E(G) thì v_1 và v_2 gọi là lân cận của nhau.

Một đường đi từ đỉnh v_p đến đỉnh v_q trong đồ thị G là một dãy các đỉnh v_p, v_{i1}, v_{i2}, v_{in},..., v_q mà (v_p, v_{i1}), (v_{i1}, v_{i2})... (v_{in}, v_q) là các cung trong E(G). Số lượng các cung trên đường đi ấy gọi là độ dài của đường đi.

Thí dụ: trong hình 6.1a: 1, 3, 4 là đường đi từ đỉnh 1 đến đỉnh 4 nó có độ dài bằng 2.

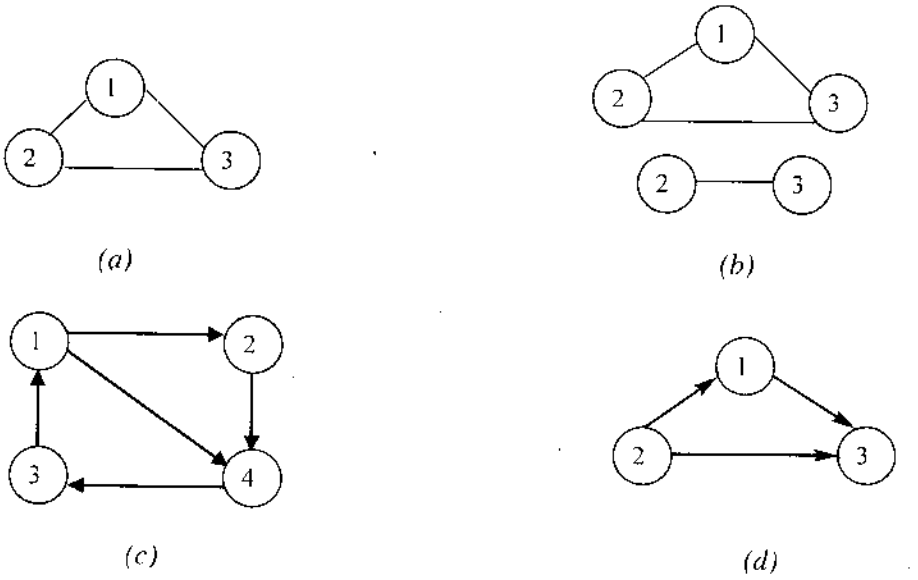
Hình 6.1b: 1, 3, 5, 2, 4 có thể là đường đi từ đỉnh 1 đến đỉnh 4, nó có độ dài bằng 4.

- Một đường đi đơn là đường đi mà mọi đỉnh trên đó, trừ đỉnh đầu và đỉnh cuối đều khác nhau.

- Một chu trình là một đường đi đơn mà đỉnh đầu và đỉnh cuối trùng nhau. Trong hình 6.1a đường đi 1, 3, 4, 1 là một chu trình.

- Trong đồ thị G hai đỉnh v_i và v_j gọi là liên thông nếu có một đường đi từ v_i đến v_j.

- G gọi là liên thông nếu mọi cặp đỉnh phân biệt v_i, v_j trong V(G) đều có đường đi từ v_i tới v_j. Chẳng hạn như:



Hình 6.2

a) Đồ thị không định hướng liên thông; b) Đồ thị không định hướng không liên thông; c) Đồ thị định hướng liên thông; d) Đồ thị định hướng không liên thông.

Ở mỗi cung có thể gắn một giá trị thể hiện một thông tin nào đó liên quan tới cung gọi là trọng số (đồ thị có trọng số). Ví dụ trong đồ thị mô tả mạng lưới giao thông giữa các thành phố trọng số có thể là khoảng cách giữa các thành phố.

II. BIỂU DIỄN ĐỒ THỊ

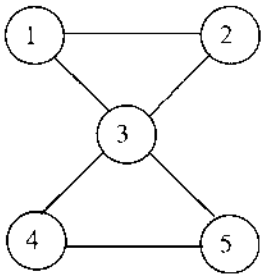
1. Biểu diễn bằng ma trận lân cận (ma trận kề)

Xét một đồ thị G(V, E) với v gồm có n đỉnh ($n \geq 1$). Giả sử các đỉnh được đánh số theo một quy định nào đó. Ma trận A biểu diễn G là một ma trận vuông kích thước n x n. Các phần tử của ma trận có giá trị 0 và 1. Nếu phần tử $A_{ij} = 1$ thì tồn tại một cung (v_i, v_j) trong E; nếu $A_{ij} = 0$ thì không tồn tại cung đó.

Với đồ thị không định hướng thì ma trận lân cận biểu diễn nó có các phần tử đối xứng qua đường chéo chính, nghĩa là nếu có phần tử bằng 1 ở hàng i cột j thì cũng có một phần tử bằng 1 ở hàng j cột i. Với đồ thị định hướng thì không như vậy.

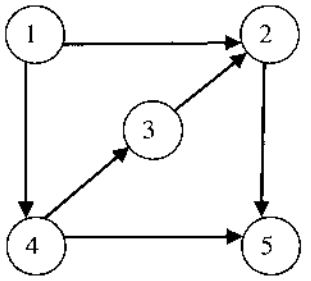
Ví dụ:

a)



	1	2	3	4	5
1	0	1	1	0	0
2	1	0	1	0	0
3	1	1	0	1	0
4	0	0	1	0	1
5	0	0	1	1	0

b)



	1	2	3	4	5
1	0	1	0	1	0
2	0	0	0	0	1
3	0	1	0	0	0
4	0	0	1	0	1
5	0	0	0	0	0

Với cách đánh số khác nhau sẽ có các ma trận lân cận khác nhau của cùng một đồ thị G. Tuy nhiên một ma trận nào đó của G cũng có thể được lập từ một ma trận lân cận khác của G bằng cách đổi chỗ một số hàng và cột tương ứng. Vì vậy thứ tự ấn định cho các nút có thể tuỳ ý.

Với đồ thị có trọng số thì ma trận lân cận có thể được lập bằng cách thay giá trị 1 bởi trọng số.

2. Biểu diễn bằng danh sách lân cận

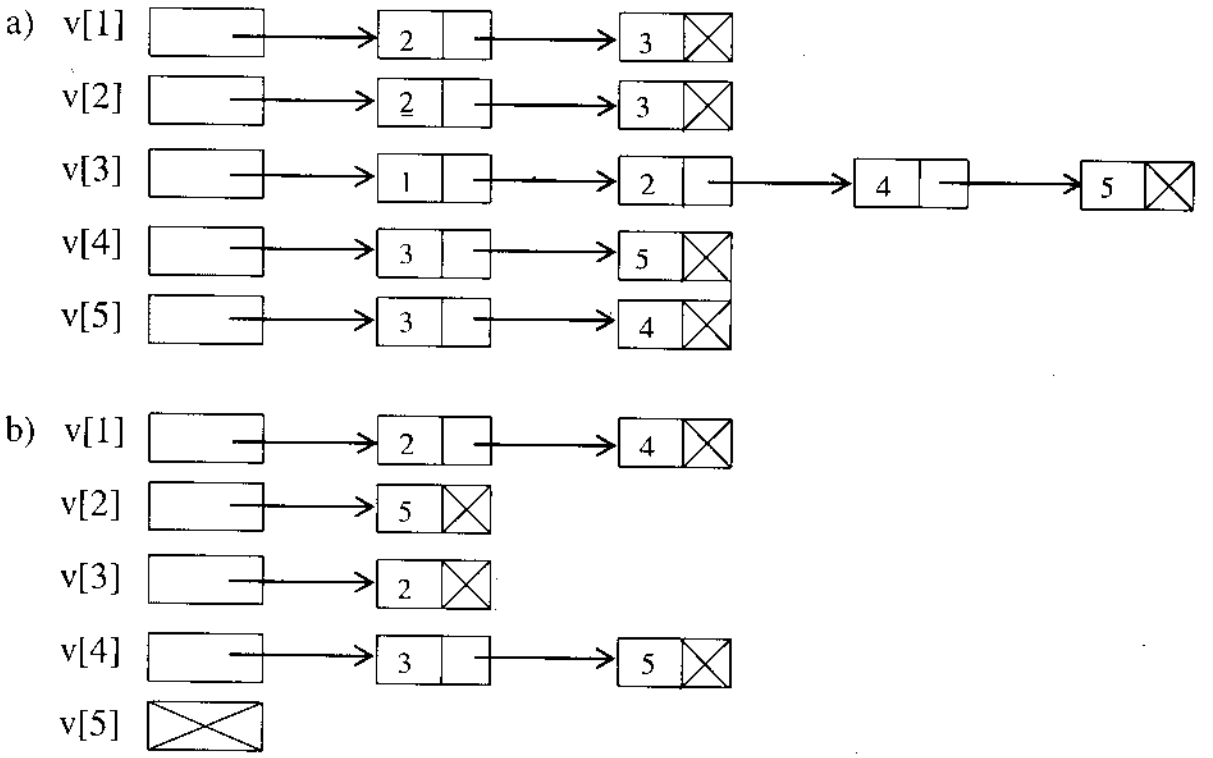
n hàng của ma trận thay bởi n danh sách móc nối.

Với mỗi đỉnh G có một danh sách tương ứng. Các nút trong danh sách i biểu diễn các đỉnh lân cận của nút i. Mỗi nút có hai trường: VERTEX và LINK.

Trường VERTEX chứa chỉ số (số thứ tự) của các đỉnh lân cận của đỉnh i.

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

Với đồ thị trên biểu diễn như sau:



Mỗi danh sách có một nút "Đầu danh sách". Các nút đầu này được tổ chức kế tiếp (dưới dạng vectơ) để truy nhập được nhanh.

Trường hợp đồ thị không định hướng có n đỉnh, e cung thì cần n nút đầu và 2e "nút danh sách".

Với đồ thị định hướng thì số nút danh sách chỉ cần e.

III. PHÉP DUYỆT MỘT ĐỒ THỊ

Xét một đồ thị không định hướng G(V, E) và một đỉnh v trong V(G), ta cần thăm tất cả các đỉnh thuộc G mà có thể *"với tới"* được từ đỉnh v (nghĩa là thăm mọi nút liên thông với v).

Có hai cách để giải quyết như sau:

- Phép *tìm kiếm theo chiều sâu* (Depth First Search - DFS)
- Phép *tìm theo chiều rộng* (Breadth First Search - BFS)

1. Tìm kiếm theo chiều sâu (DFS)

Tìm kiếm theo chiều sâu với một đồ thị không định hướng được thực hiện như sau:

Xuất phát từ đỉnh v được thăm, tiếp đó là đỉnh ω chưa được thăm là lân cận của v sẽ được thăm và một phép tìm kiếm theo chiều sâu xuất phát từ ω được thực hiện.

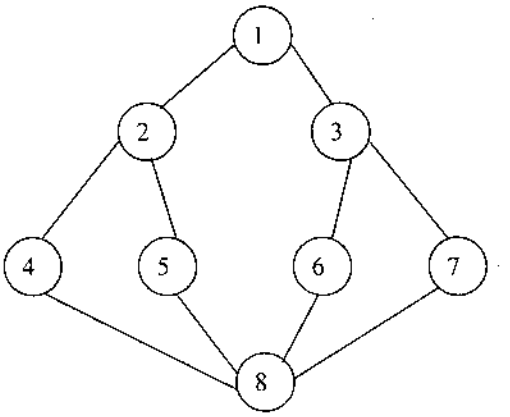
Khi một đỉnh u đã được *"với tới"* mà mọi đỉnh lân cận của nó đã được thăm rồi thì sẽ quay ngược lên đỉnh cuối cùng vừa được thăm (mà còn có đỉnh ω lân cận với nó chưa được thăm) và một phép tìm kiếm theo chiều sâu xuất phát từ ω lại được thực hiện. Phép tìm kiếm sẽ kết thúc khi không còn một nút nào chưa được thăm mà vẫn có thể với tới được từ một nút đã được thăm.

Ví dụ:

Với đồ thị như hình 6.3

Xuất phát từ đỉnh v1 ta có dãy các đỉnh được thăm như sau:

v1, v2, v4, v8, v9, v5, v6, v3, v7, v10



Hình 6.3

Giải thuật của phép duyệt này có thể được xây dựng như sau:

Procedure DFS(v);

{Cho một đồ thị không định hướng G(V, E) gồm n đỉnh và 1 vectơ *Tham(n)* gồm *n* phần tử bắt đầu bằng 0. Giải thuật này thực hiện phép thăm mọi đỉnh *"với tới"* được từ đỉnh v}

1. Tham(v):=1 {*Tham* dùng để đánh dấu các đỉnh đã được thăm}
2. **for** mỗi đỉnh ω lân cận của v **do**
   if Tham(w) = 0 **then** Call DFS(ω);
3. **return**

** Thời gian thực hiện giải thuật*

Trong trường hợp G được biểu diễn bởi một danh sách lân cận thì đỉnh ω lân cận của v sẽ được xác định bằng cách dựa vào danh sách móc nối ứng với v. Vì giải thuật DFS chỉ xem xét mỗi nút trong danh sách lân cận nhiều nhất một lần mà lại có 2e nút danh sách (ứng với e cung), nên thời gian để hoàn thành phép tìm kiếm chỉ là O(e).

Còn nếu G được biểu diễn bởi ma trận lân cận thì thời gian để xác định mọi đỉnh lân cận của v là O(n). Vì tối đa có n đỉnh được thăm, nên thời gian tìm kiếm tổng quát sẽ là $O(n^2)$.

Với giải thuật DFS ta có thể xác định được G liên thông hay không, hoặc tìm được các bộ phận liên thông của G nếu G không liên thông.

2. Tìm kiếm theo chiều rộng (BFS)

Đỉnh xuất phát v được thăm đầu tiên, sau đó các đỉnh chưa được thăm mà là lân cận của v sẽ được thăm kế tiếp theo nhau, rồi đến các đỉnh chưa được thăm là các đỉnh lân cận của các đỉnh này và cứ tiếp tục như vậy.

Ví dụ như đối với đồ thị đã xét ở hình 6.3 xuất phát từ đỉnh v1 thì thứ tự các đỉnh được thăm là: v1, v2, v3, v4, v5, v6, v7, v8.

Giải thuật của phép duyệt này như sau:

Procedure BFS(v);

{Bắt đầu từ v, các đỉnh được thăm sẽ được đánh dấu Tham(v):=1. Ban đầu tất cả các đỉnh đều chưa được thăm cho nên các Tham(v) đều bằng 0. Giải thuật này sử dụng một Queue với 2 con trỏ F và R trỏ tới lối trước và lối sau và các thủ tục bổ sung, loại bỏ một phần tử.}

```
1. Tham(v):=1;
2. Tạo Queue và nạp v vào
3. While Q không rỗng do begin
   Call Loaibo(v,Q);      {Lấy v ra khỏi Q}
   for các đỉnh w lân cận với v do
         if Tham(w):=0 then
               begin
                     Call Bosung(w,Q);
                     Tham(w):=1;
               end;
                           end;
4. Return
```

Mỗi đỉnh được thăm sẽ được nạp vào Queue chỉ một lần vì vậy câu lệnh **while** lặp lại nhiều nhất n lần.

Nếu G được biểu diễn bởi ma trận lân cận thì câu lệnh **for** sẽ chi phí O(n) thời gian với mỗi đỉnh, do đó thời gian chi phí toàn bộ sẽ là $O(n^2)$.

Trong trường hợp G được biểu diễn bởi danh sách lân cận thì chi phí tổng quát cũng là O(e).

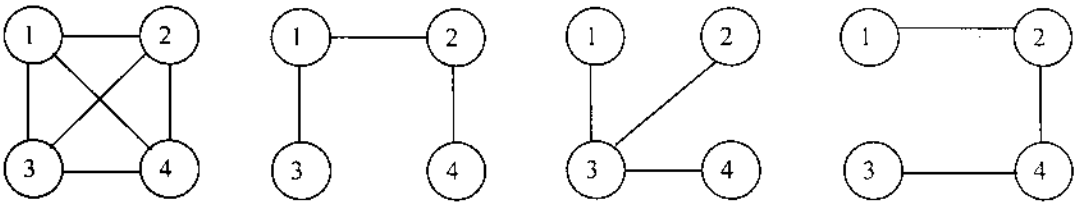
IV. CÂY KHUNG VÀ CÂY KHUNG VỚI GIÁ CỰC TIỂU

Khi đồ thị G liên thông thì với một phép tìm kiếm bất kỳ, bắt đầu từ một đỉnh bất kỳ, ta đều có thể thăm được mọi đỉnh của G. Trong trường hợp này các cung của G được phân thành 2 tập:

- Tập T gồm các cung được duyệt.
- Tập B gồm các cung còn lại.

Tất cả các cung trong T và các đỉnh tương ứng tạo thành một cây bao gồm mọi đỉnh (nút) của G. Một cây như vậy gọi là *cây khung* của G.

Sau đây là một đồ thị và ba cây khung tương ứng với nó:

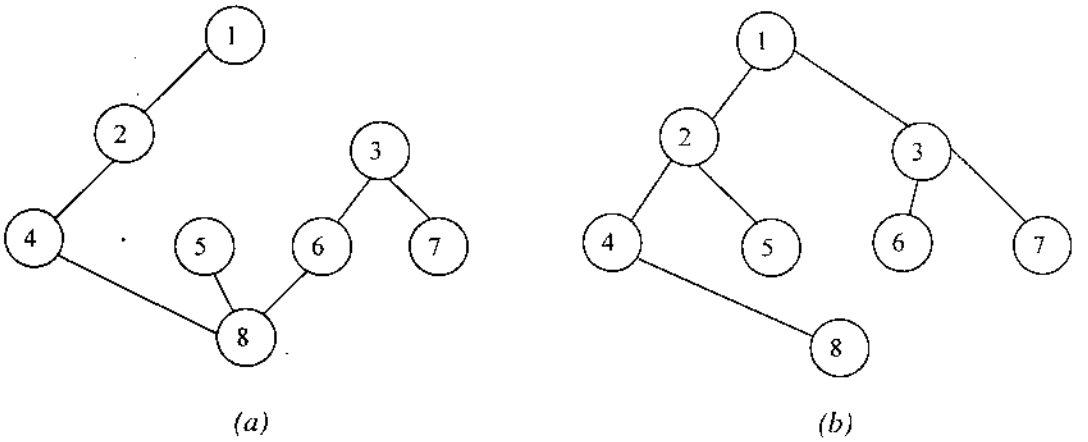


Hình 6.4

Tuỳ theo phép tìm kiếm theo chiều sâu hay chiều rộng được gọi là:

- Cây khung theo chiều sâu hay cây khung theo chiều rộng.

Với đồ thị hình 6.3 ta được:



Hình 6.5

a) Cây khung theo chiều sâu; b) Cây khung theo chiều rộng

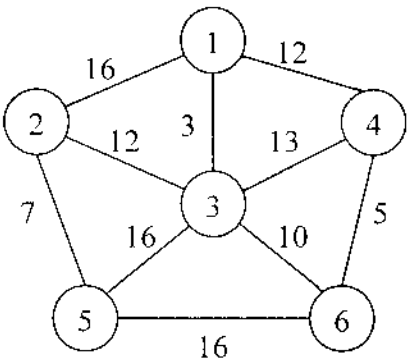
Ta nhận thấy nếu một cung bất kỳ (v, w) nào của B được bổ sung vào cây khung T thì một chu trình mới sẽ xuất hiện. Chu trình mới này bao gồm cung (v, w) và mọi cung trên đường đi từ w tới v trong T.

Sau đây ta xét một ứng dụng của cây khung, đó là việc xác định cây khung với giá cực tiểu của một đồ thị liên thông có trọng số. Giá của một cây khung là tổng các trọng số ứng với các cung của cây ấy. Bài toán cây khung với giá cực tiểu là một trong những bài toán ứng dụng quan trọng trong thực tế. Chẳng hạn, nếu các đỉnh của G biểu diễn các thành phố, các cung biểu diễn đường nối giữa hai thành phố và trọng số biểu diễn giá tiền xây dựng hoặc độ dài của đường nối... thì việc chọn một cây khung với giá cực tiểu là tương ứng với việc xây dựng một mạng đường giao thông có khả năng liên thông với mọi thành phố với giá xây dựng tổng cộng là ít nhất hoặc với tổng độ dài là ngắn nhất...

Có nhiều giải thuật khác nhau để xây dựng cây khung với giá trị cực tiểu. Sau đây ta xét hai giải thuật tiêu biểu là giải thuật Kruskal và giải thuật Prim.

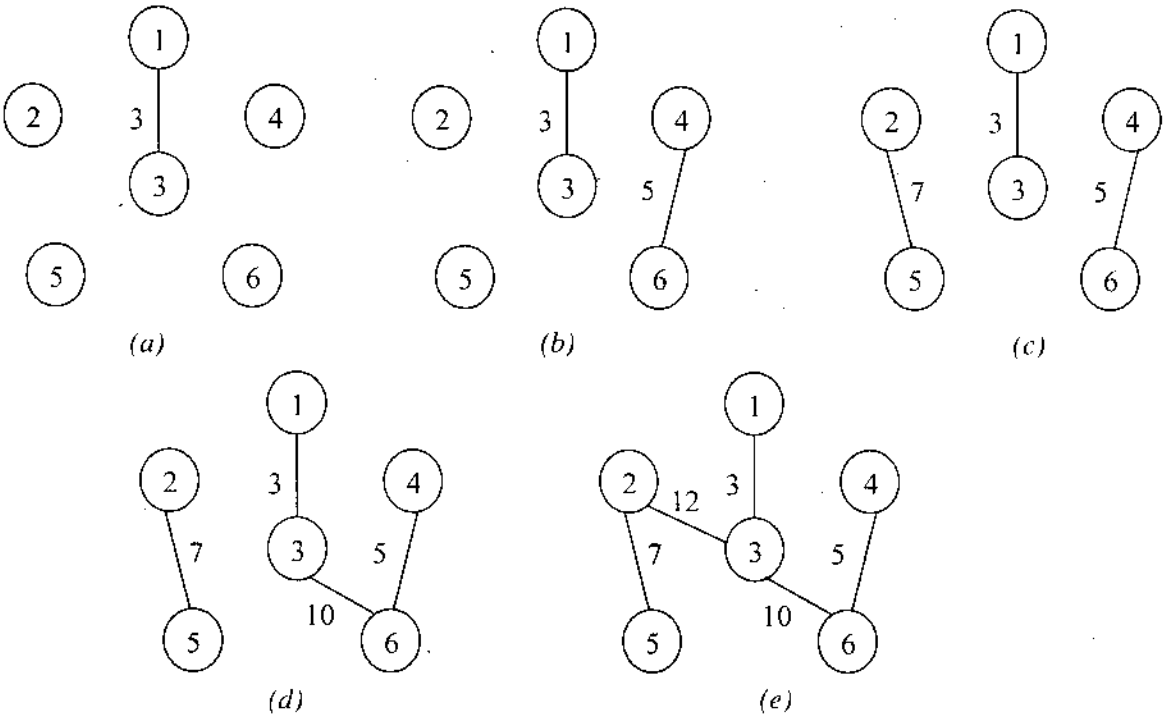
1. Giải thuật Kruskal

Cây khung với giá cực tiểu T sẽ được xây dựng dần dần từng cung một. Các cung được xét để đưa vào T dựa theo thứ tự không giảm của giá tương ứng với chúng. Một cung được đưa vào T nếu nó không tạo nên một chu trình với các cung đã có trong T. Vì G liên thông với số đỉnh n>0, nên sẽ có n-1 cung được chọn để đưa vào T.



Hình 6.6

Ví dụ: Với đồ thị đã cho ở hình 6.6, cây khung với giá cực tiểu được xây dựng bởi giải thuật Kruskal được thể hiện từng bước như sau:



Hình 6.7

Các cung của đồ thị được xét để đưa vào cây khung với giá cực tiểu theo thứ tự: (1, 3), (4, 6), (2, 5), (3, 6), (2, 3). Còn các cung khác bị loại vì nếu đưa thêm bất kỳ một cung khác vào trong T thì nó sẽ nối 2 đỉnh trong T tạo thành một chu trình.

Cây khung có 5 cành và có giá trị bằng 37. Đây chính là cây khung với giá cực tiểu.

Giải thuật Kruskal được diễn tả như sau:

```
1. T:=Φ    {Ban đầu T rỗng}
2. While  T chứa ít hơn (n-1) cung do
3.        Begin chọn một cung (v, ω) từ E có giá trị thấp nhất
4.              loại (v, ω) khỏi E.
5.              if (v, ω) không tạo nên một chu trình trong T
6.                    then đưa (v, ω) vào T
7.               else loại bỏ(v, ω)
```

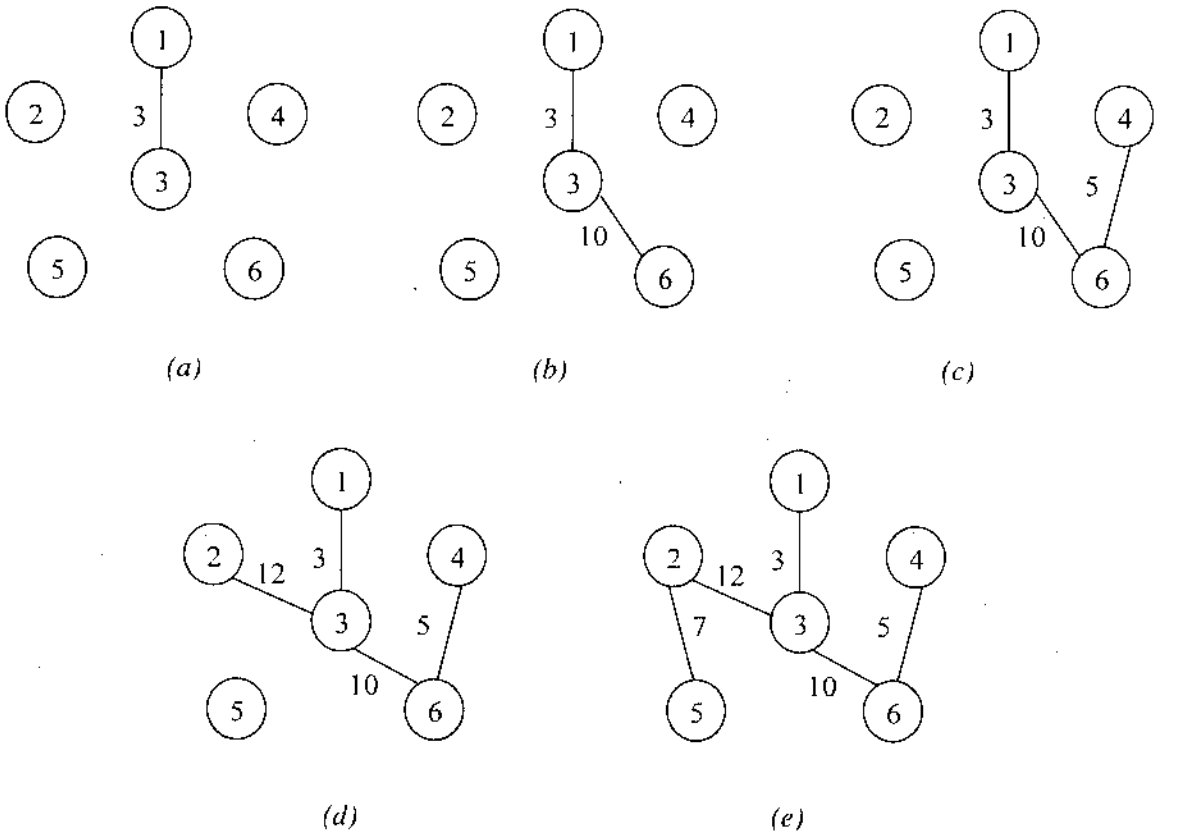
End

8. **Return**.

2. Giải thuật Prim

Giải thuật Prim về cơ bản có phương thức giải thuật tương tự như giải thuật Kruskal nhưng có điểm khác nhau ở chỗ: Khi một cung được xét để đưa vào T nếu nó không tạo nên một chu trình với các cung đã có trong T và phải liên thuộc với T. Vì G liên thông với số đỉnh n>0, nên sẽ có n-1 cung được chọn để đưa vào T.

Ví dụ: Với đồ thị đã cho ở hình 6.6, cây khung với giá cực tiểu được xây dựng bởi giải thuật Prim được thể hiện từng bước như sau:



Hình 6.8

Các cung của đồ thị được xét để đưa vào cây khung với giá cực tiểu lần lượt theo thứ tự: (1, 3), (3, 6), (6, 4), (3, 2), (2, 5).

Cây khung có 5 cành và có giá trị bằng 37. Đây chính là cây khung với giá cực tiểu.

Giải thuật Prim được diễn tả như sau:

1. T:=Φ {Ban đầu T rỗng}
2. **While** T chứa ít hơn (n-1) cung **do**
3. **Begin** chọn một cung (v, ω) từ E có giá trị thấp nhất
4. **if** (v, ω) không tạo nên một chu trình trong T **and** (v,w) liên thuộc với T **then begin**
   đưa (v, ω) vào T;
   loại (v, ω) khỏi E;
   end;

End

5. **Return.**

* Thời gian thực hiện giải thuật của cả hai thuật toán là như nhau O(e*log(e)) với e là số các cung thuộc E(G).

V. ÁP DỤNG

Đồ thị thường được dùng để biểu diễn các bài toán trong thực tế như mạng giao thông, mạng máy tính, một số bài toán tối ưu .v.v. Nếu được gán thêm trọng số vào các cung thì các trọng số đó sẽ mang ý nghĩa thực tiễn riêng trong từng trường hợp riêng biệt như đã nêu ở phần trên. Đối với các bài toán thực tế thì thường phải trả lời hai câu hỏi sau:

1) Có tồn tại một đường đi từ A → B hay không?

2) Nếu có nhiều đường đi từ A → B thì đường nào là ngắn nhất?

Để trả lời các câu hỏi này chúng ta xét một số bài toán sau:

1. Bài toán bao đóng truyền ứng

Bài toán 1 nêu trên có thể được giải quyết dễ dàng bằng cách sử dụng ma trận lân cận của đồ thị.

Có thể coi ma trận lân cận như một trận dạng Boolean và có thể tác động lên ma trận các phép toán logic:

- Phép cộng (phép hoặc): ∨
- Phép nhân (phép và): ∧

a	b	a ∨ b	a ∧ b
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

Đối với 2 ma trận Bool A và B, kích thước n x n:
Phép cộng hai ma trận để cho ra ma trận tổng C

$$C = A \vee B$$

Được thực hiện như sau:

$$C_{ij} = a_{ij} \vee b_{ij} \quad \text{với} \quad \begin{cases} 1 \le i \le n \\ 1 \le j \le n \end{cases}$$

Phép nhân hai ma trận để cho ra ma trận tích D được thực hiện bằng cách

$$d_{ij} = \bigvee_{k=1}^{n} \left(a_{ik} \wedge b_{kj} \right) \qquad \text{với} \qquad \begin{cases} 1 \le i \le n \\ 1 \le j \le n \end{cases}$$

Ta biết rằng với ma trận lân cận A biểu diễn đồ thị, nếu $a_{ij} = 1$ thì nghĩa là tồn tại một cung đi từ $i \rightarrow j$.

Nếu ma trận $A^{(2)} = A \wedge A$ có phần tử ở hàng i cột j bằng 1 thì ít nhất có một đường đi có độ dài 2 từ đỉnh i tới đỉnh j, vì:

$$a_{ij}^{(2)} = \bigvee_{k=1}^{n} \left(a_{ik} \wedge a_{kj} \right)$$

Từ đó suy ra:

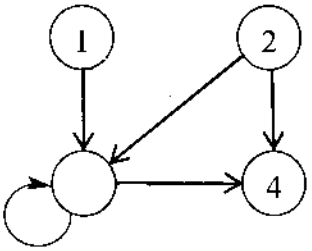
$A^{(r)} = A \wedge A^{(r-1)}$ với r = 3, 4,....

$a_{ij}^{(r)} = 1$ thì ít nhất có một đường đi độ dài là r đi từ đỉnh $i \rightarrow j$

Như vậy, nếu lập ma trận: $P = A \vee A^2 \vee \ldots \vee A^n = \bigvee_{k=1}^{n} A^k$

Thì P sẽ cho biết: có hay không một đường đi, độ dài lớn nhất là n đi từ $i \rightarrow j$.
P được gọi là *ma trận đường đi* (Path matrice) của đồ thị G.

Ví dụ: Với đồ thị



Hình 6.9

Việc tính P có thể dễ dàng thực hiện bởi giải thuật Warshall

Procedure Warshall **(A,P,n)**;

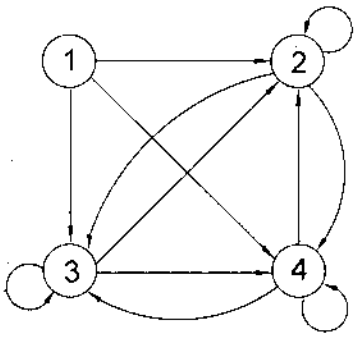
1. P:= A {Ma trận ban đầu}
2. **For** k:=1 **to** n **do**
   For i:=1 **to** n **do**
   For j:=1 **to** n **do**
   P[i,j]:= P[i,j] or P[i,k] and P[k,j]
3. **Return**.

Với đồ thị hình 6.9 quá trình tính P mô tả như sau:

$$\begin{bmatrix}0&0&1&0\\0&0&1&0\\0&0&1&1\\0&1&0&0\end{bmatrix} \quad \begin{bmatrix}0&0&1&1\\0&0&1&1\\0&1&1&1\\0&0&1&0\end{bmatrix} \quad \begin{bmatrix}0&1&1&1\\0&1&1&1\\0&1&1&1\\0&0&1&1\end{bmatrix} \quad \begin{bmatrix}0&1&1&1\\0&1&1&1\\0&1&1&1\\0&1&1&1\end{bmatrix} \quad \begin{bmatrix}0&1&1&1\\0&1&1&1\\0&1&1&1\\0&1&1&1\end{bmatrix}$$

$A \qquad A^{(2)} \qquad A^{(3)} \qquad A^{(4)} \qquad P$

Với ma trận P đồ thị tương ứng như sau:



Hình 6.10

Một đồ thị như vậy gọi là bao đóng truyền ứng của G.

Chú ý:

1. Ma trận đường đi chỉ cho ta biết có hay không ít nhất một đường đi tại một cặp đỉnh hoặc một chu trình tại một cặp đỉnh nào đó, chứ nó không chỉ ra mọi con đường có thể có đó, và cũng không cho ta biết đường nào ngắn nhất.

2. Nếu chỉ quan tâm đến việc có hay không một đường đi từ đỉnh i tới đỉnh j. Chỉ cần tính ma trận dưới đây là đủ.

$$P_{n-1} = A \vee A^{(2)} \; A^{(3)} \; .... \; A^{(n-1)}$$

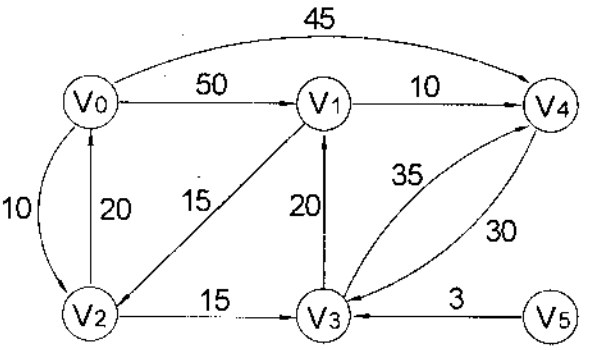
Vì với n đỉnh thì một đường đi từ một đỉnh i đến đỉnh j có độ dài n không phải là một đường đi đơn.

2. Bài toán một nguồn, mọi đích

Cho một đồ thị G(V, E) và một hàm trọng số w(e) cho các cung e của G và một đỉnh nguồn v_0.

Bài toán đặt ra là: Xác định đường đi ngắn nhất từ v_0 đến mọi đỉnh của G. Với độ dài đường đi là tổng các trọng số tương ứng với các cung trên đường đi và các trọng số đều là số dương.

Ví dụ:



Hình 6.11

Nếu v_0 là nguồn thì đường đi ngắn nhất từ $v_0 \rightarrow v_1, v_2, v_3, v_4, v_5$ là:

Đường đi	Độ dài
v_0v_2	10
$v_0v_2v_3$	25
$v_0v_2v_3v_1$	45
v_0v_4	45

Không có đường đi tới v_5.

** Giải thuật tìm đường đi ngắn nhất*

Gọi S là tập các đỉnh (kể cả v_0) theo đó các đường đi ngắn nhất đã được xác lập.

Đối với 1 đỉnh ω không thuộc S, gọi DIST(ω) là độ dài đường đi ngắn nhất từ v_0, qua các đỉnh, chỉ trong S và kết thúc ở ω.

Ta có một số nhận xét như sau:

1) Nếu đường đi ngắn nhất tiếp theo hướng tới 1 đỉnh ω, thì đường đi đó bắt đầu từ v_0, kết thúc ở ω và chỉ qua những đỉnh đã thuộc S.

2) Đích của đường đi sinh ra tiếp theo phải là một đỉnh ω nào đó hiện chưa thuộc S mà có DIST(ω) ngắn nhất so với giá trị của DIST ứng với mọi đỉnh khác cũng đang không thuộc S.

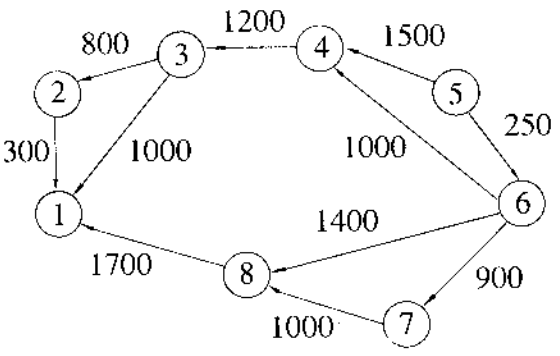
3) Nếu chọn được một đỉnh ω như trong (2) và đã sinh ra được 1 đường đi ngắn nhất từ v_0 tới ω thì sẽ trở thành 1 phần tử của S.

Như vậy độ dài đường đi ngắn nhất xuất phát từ v_0 đi qua các đỉnh thuộc S và kết thúc ở đỉnh ω không thuộc S, có khả năng sẽ giảm đi; nghĩa là giá trị DIST(ω) có thể thay đổi. Nếu nó thay đổi thì nó phải được lập từ một đường đi ngắn hơn, bắt đầu từ v_0 đến u và từ u tới ω tất cả phải thuộc S. Hơn nữa đường đi từ v_0 tới u phải là đường đi ngắn nhất vì nếu không thì DIST(ω) sẽ xác định không đúng. Đường đi từ u tới ω phải chọn sao cho nó không chứa 1 đỉnh trung gian nào.

Vậy DIST(ω) sẽ thay đổi vì đường đi từ v_0 tới u rồi tới ω đã có đoạn đường đi từ v_0 tới u là đường đi ngắn nhất và đường đi từ u tới chỉ là cung (u, ω). Độ dài đường đi đó là: DIST(u) + độ dài (u, ω).

Giải thuật đường đi ngắn nhất dựa theo quan điểm này được đưa ra đầu tiên bởi Dijkstra.

Giả thiết n đỉnh của G được đánh số từ 1 đến n. Tập S được thể hiện dưới dạng vectơ bit S[i]=0 nếu $i \notin S$ và S[i]=1 nếu $i \in S$ còn độ thị có trọng số được biểu diễn bởi ma trận lân cận COST với phần tử COST[i,j] là trọng số của cung (i,j), COST[i,j]= ∞ nếu cung (i,j) không có, COST[i,j]=0 nếu i=j.



Hình 6.12

Với đồ thị hình 6.12 thì ma trận COST có dạng:

	1	2	3	4	5	6	7	8
1	0							
2	300	0						
3	1000	800	0			$+\infty$		
4			1200	0				
5				1500	0	250		
6				1000		0	900	1400
7		$+\infty$					0	1000
8	1700							0

Giải thuật tìm đường đi ngắn nhất như sau:

Procedure SORTEST_PATH(v, COST, DIST, n)

{DIST(i), $1 \leq j \leq n$: Thể hiện độ dài đường đi ngắn nhất từ đỉnh v đến đỉnh j trong đồ thị định hướng G có n đỉnh, DIST(v)= 0. G được biểu diễn bởi ma trận lân cận COST kích thước n x n}

1. **For** i:=1 **to** n **do**
   begin
   S[i]:=0; DIST[i]:=COST[v,i]
   end;
2. S[v]:=1; DIST[v]:=0; k=2 {đưa v vào S}
3. **While** k < n **do** {xác định n-1 đường đi từ đỉnh v}
4. **Begin**
   chọn u sao cho DIST[u]= min (DIST[u]);
5. S[u]:=1; k:=k+1; {đưa u vào tập S}
6. **For** ∀ω với S[ω]=0 **do**
7. DIST[ω]:= min(DIST[ω], DIST[u] + COST[u,ω])

   End
8. **Return**

Với đồ thị hở hình 6.12 và giả sử v là nút 5 thì diễn biến của tác động giải thuật SORTEST_PATH được thể hiện qua bảng sau:

Bước lặp	S	Đỉnh được chọn	DIST	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
Khởi đầu	-			∞	∞	∞	1500	0	250	∞	∞
1	5	6		∞	∞	∞	1250	0	250	1150	1650
2	5,6	7		∞	∞	∞	1250	0	250	1150	1650
3	5,6,7	4		∞	∞	2450	1250	0	250	1150	1650
4	5,6,7,4	8		3350	∞	2450	1250	0	250	1150	1650
5	5,6,7,4,8	3		3350	3250	2450	1250	0	250	1150	1650
6	5,6,7,4,8,3	2		3350	3250	2450	1250	0	250	1150	1650
	5,6,7,4,8,3,2										

3. Bài toán sắp xếp Topo

Thông thường một công việc lớn bao giờ ta cũng phải chia nó thành từng phần việc nhỏ để thực hiện. Có những phần việc có thể được thực hiện độc lập nhưng cũng có những phần việc chỉ thực hiện được khi một số phần việc khác đã được làm xong.

Chẳng hạn để xây một ngôi nhà cần phải hoàn thành những công việc sau:

Số hiệu	Tên công việc	Việc cần làm trước
M1	Đào móng	Không
M2	Xây móng	M1
M3	Xây tường	M2
M4	Làm đồ gỗ	Không
M5	Lắp cửa	M3, M4
M6	Đặt mái	M3, M4
M7	Lợp mái	M6
M8	Trát tường	M3, M7
M9	Đặt CT phụ	M7
M10	Quét vôi	M7, M8

Như vậy những công việc "đào móng" hay "làm đồ gỗ" có thể hoàn thành độc lập. Nhưng có những việc như "lắp cửa" sẽ không thể bắt đầu nếu chưa "xây tường" và chưa "làm đồ gỗ". Như vậy giữa quan hệ M5 và M3, M4 có một quan

hệ "được làm trước". M3, M4 được làm trước M5. Quan hệ đó được gọi là một *"thứ tự bộ phận"*. Đó là một quan hệ giữa các phần tử của một tập S, được ký hiệu là "$\prec$" đọc là *"đứng trước"* và quan hệ đó thoả mãn tính chất sau đối với các phần tử phân biệt x, y, z của S:

1) Nếu x $\prec$ y và y $\prec$ z thì x $\prec$ z (tính bắc cầu)

2) Nếu x $\prec$ y thì không có y $\prec$ x (tính phản xứng)

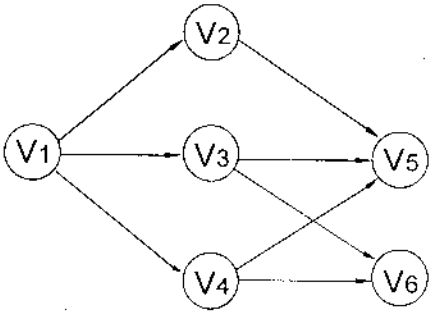
3) Không có x $\prec$ x (tính không phản xạ)

Do ý nghĩa thực tế ta luôn giả thiết S là một tập hữu hạn. Một thứ tự bộ phận có thể minh hoạ bằng một đồ thị định hướng trong đó các đỉnh ứng với các phần tử. Các cung ứng với các quan hệ thứ tự.

Vấn đề đặt ra là làm sao sắp xếp các đỉnh của đồ thị theo một thứ tự tuyến tính để sao cho khi xét tới một đỉnh nào đó, thì các đỉnh trước đó đã được xét rồi (đã hoàn thành). Một thứ tự tuyến tính với đặc điểm như vậy gọi là *thứ tự Topo* và cách sắp xếp một tập đối tượng có thứ tự bộ phận thành thứ tự Topo được gọi là *sắp xếp Topo*.

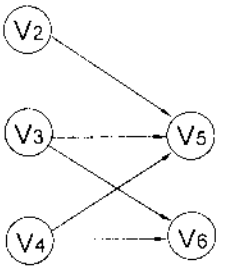
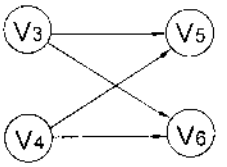
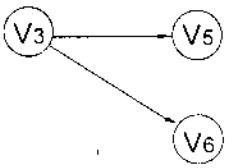
Như vậy, đối với một đồ thị định hướng không có chu trình (do tính chất của thứ tự bộ phận) thì sắp xếp Topo là một quá trình định ra một thứ tự tuyến tính liên quan đến các đỉnh của đồ thị ấy, sao cho nếu có một cung từ đỉnh i đến đỉnh j, thì i cũng xuất hiện trước j trong thứ tự tuyến tính. Trong thực tế có những yêu cầu liên quan tới sắp xếp Topo, sắp xếp Topo có liên quan tới kỹ thuật PERT dùng trong lý thuyết quy hoạch.

Tư tưởng của giải thuật: Ta bắt đầu chọn một cung mà không có cung nào đi tới nó (không có đỉnh nào "trước" nó), đỉnh này sẽ được đưa ra. Sau đó nó và các cung xuất phát từ nó sẽ bị loại khỏi đồ thị, quá trình được lặp lại với phần còn lại của đồ thị cho tới khi các đỉnh đó được chọn ra hết. Nếu cùng một lúc có nhiều đỉnh đều không có cung đi tới nó, thì việc chọn một đỉnh nào đó trong số ấy là tuỳ ý.



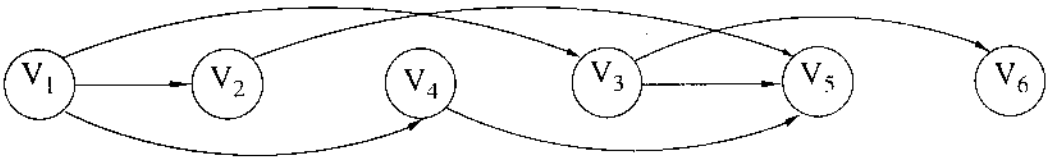
Hình 6.13

Ví dụ: Với đồ thị hình 6.13 thì các bước thực hiện sắp xếp topo được thể hiện theo hình 6.14.

Đỉnh được đưa ra	Phần đồ thị còn lại
V1	
V2	
V4	
V3	
V5	
V6	

Hình 6.14

Như vậy dãy đưa ra là V_1, V_2, V_4, V_3, V_5, V_6 mà dưới dạng đồ thị có thể biểu diễn:



Hình 6.15

Với giải thuật này, đồ thị coi như được biểu diễn dưới dạng danh sách lân cận. Tuy nhiên nút đầu danh sách sẽ có hai trường: COUNT và LINK.

Trường COUNT chứa số đếm các cung đi đến đỉnh đó (số đỉnh trước nó).

Trường LINK chứa con trỏ, trỏ tới nút đầu tiên của danh sách lân cận tương ứng với mỗi đỉnh.

Danh sách lân cận của đồ thị này coi như đã được tạo lập xong trước khi thực hiện giải thuật. Danh sách các đỉnh với số đếm COUNT bằng 0, sẽ được tổ chức dưới dạng Stack. Để tiết kiệm người ta dùng luôn trường COUNT ở nút đầu danh sách (ứng với đỉnh mà COUNT=0) để làm trường móc nối của Stack, vì khi đó trường này không còn cần dùng nữa.

Giải thuật như sau:

```
Procedure TOPO_ORDER(COUNT, VERTEXT, LINK, n)
{Trong giải thuật này Top là con trỏ trỏ tới đỉnh Stack}
1.{Khởi tạo Stack}
   Top:=0;
2.{Tạo lập Stack của các đỉnh không có đỉnh trước nó}
   for i:= 1 to n do
3.       if  COUNT(i)=0 then begin
                 COUNT(i):=Top;
                 Top:=i;
                                    end;
4.{Đưa các đỉnh ra theo thứ tự Topo}
   for i:= 1 to n do begin
5.       if Top = 0 then return;      {có chu trình trong đồ thị}
6.       j:= Top; Top:= COUNT(Top);
         Write(j);
7.       ptr:= Link(j);
8.{Giảm số đếm các đỉnh sau j}
         while ptr<>0 do begin
               k:= VERTEX(ptr);
               COUNT(k):= COUNT(k) - 1;
9.{Nạp vào Stack phần tử mới}
               if COUNT(k) = 0 then begin
```

```
                COUNT(k):= Top;
                Top:= k;
                                    end;
                            end;
10.         ptr:= Link(ptr);
                        end;
```

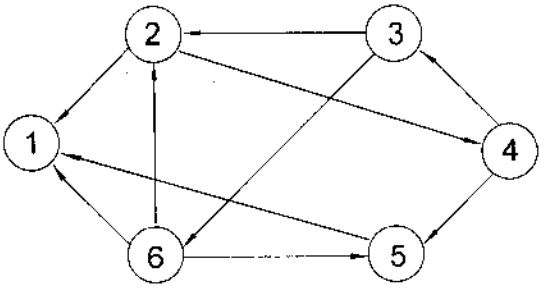
11. **Return**

Thời gian thực hiện giải thuật:

Nếu đồ thị có n đỉnh e cung thì chu trình **for** 2) 3) sẽ thực hiện với cấp thời gian là O(n), còn chu trình **while** thì với $O(d_i)$ với mỗi đỉnh i và d_i là số các cung xuất phát từ i, mà nó lại nằm trong **for** 4) nên thời gian sẽ là $O\left(\sum_{i=1}^{n} d_i\right) = O(e)$. Thời gian thực hiện giải thuật trên có cấp là O(n+e) tuyến tính với kích thước của đồ thị.

Bài tập

1. Cho đồ thị định hướng G_1

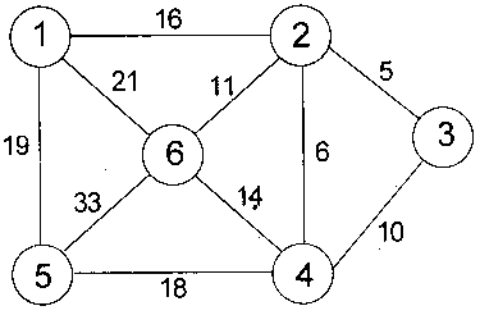


Hãy lập:

a) Ma trận lân cận của G_1

b) Danh sách lân cận của G_1

2. Cho đồ thị không định hướng G_2 có trọng số.



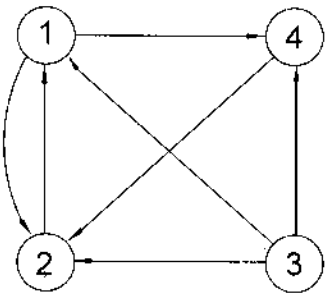
a) Hãy xây dựng cây khung theo chiều sâu của G_2.

b) Hãy xây dựng cây khung theo chiều rộng của G_2.

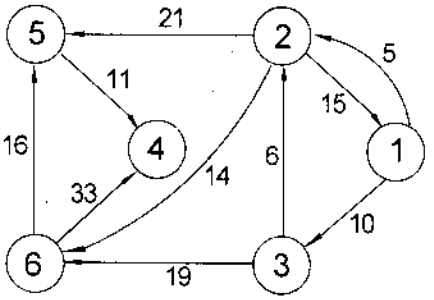
c) Dựa theo giải thuật Kruskal, nêu các bước dựng cây khung với giá cực tiểu của G_2.

d) Dựa theo giải thuật Prim, nêu các bước dựng cây khung với giá cực tiểu của G_2.

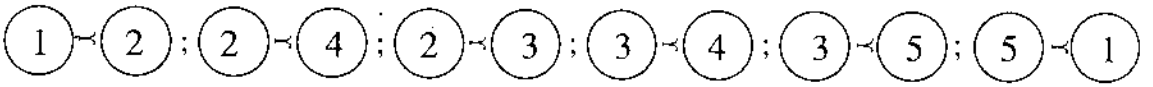
3. Cho đồ thị G_3. Hãy lập ma trận lân cận A và tính $A^{(2)}$, $A^{(3)}$, $A^{(4)}$, P. Vẽ đồ thị ứng với P.



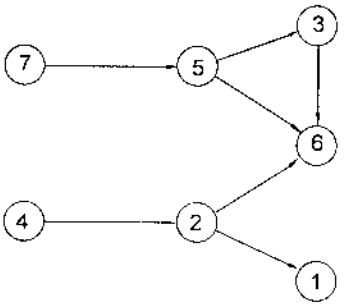
4. Cho đồ thị G_4 có trọng số. Hãy lập ma trận COST tương ứng với đồ thị và dựa vào giải thuật SHORTEST_PATH lập bảng biểu diễn các tác động của giải thuật qua các bước xác định đường đi ngắn nhất từ đỉnh 1 tới các đỉnh khác của đồ thị.



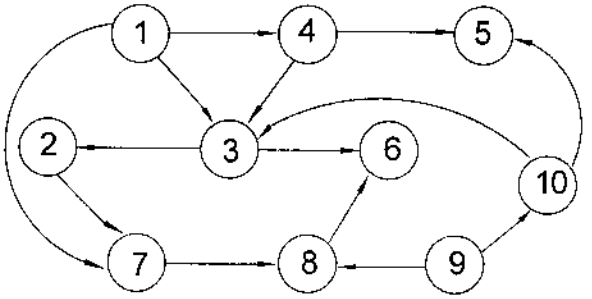
5. Quan hệ "đứng trước" ($\prec$) dưới đây có thể coi là một thứ tự bộ phận đối với các phần tử được đánh số từ 1 đến 5 hay không? Tại sao?



6. Các công việc và quan hệ thứ tự được cho bởi các đồ thị sau:



(a)



(b)

Hãy vẽ lại đồ thị với dãy các nút đã có thứ tự Topo.

Chương 7
SẮP XẾP VÀ TÌM KIẾM

Mục tiêu:

- Người học hiểu được vai trò và ý nghĩa của việc tìm kiếm và sắp xếp trong các bài toán quản lý trong thực tế cũng như các bài toán trong tin học.
- Người học biết được một số phương pháp sắp xếp đơn giản như sắp xếp kiểu lựa chọn; sắp xếp kiểu chèn (thêm dần); sắp xếp kiểu nổi bọt; sắp xếp kiểu phân đoạn;...
- Cùng với các phương pháp sắp xếp trên, người học được cung cấp một số bài toán mẫu kèm theo giải thuật minh hoạ để người học củng cố kiến thức.
- Người học có thể đánh giá và so sánh được thời gian thực hiện các giải thuật sắp xếp trên thông qua các công thức tính thời gian.
- Người học biết được một số phương pháp tìm kiếm như tìm kiếm tuần tự; tìm kiếm nhị phân hay cây nhị phân tìm kiếm.
- Chương này còn cung cấp các giải thuật tương ứng với các phương pháp tìm kiếm để minh hoạ các phương pháp tìm kiếm đó cho người học.

I. SẮP XẾP

1. Đặt vấn đề

Sắp xếp là quá trình bố trí lại các phần tử của một tập đối tượng nào đó theo một thứ tự nhất định; chẳng hạn thứ tự tăng dần (hay giảm dần) đối với một dãy số, thứ tự từ điển đối với các chữ...

Yêu cầu về sắp xếp thường xuyên xuất hiện trong các ứng dụng tin học, với những mục đích khác nhau như: sắp xếp các dữ liệu được lưu trữ trong máy tính để tìm kiếm cho thuận lợi, sắp xếp các kết quả xử lý để in ra trên bảng biểu...

Một cách tổng quát dữ liệu có thể xuất hiện dưới nhiều dạng khác nhau, nhưng ở đây ta quy ước: tập đối tượng được sắp xếp là tập các bản ghi, mỗi bản ghi bao gồm một số trường dữ liệu tương ứng với các thuộc tính khác nhau.

Trong chương này ta chỉ xét đến các phương pháp *sắp xếp trong*, nghĩa là các phương pháp tác động lên một tệp bản ghi lưu trữ đồng thời trong bộ nhớ trong, mà ta gọi là *bảng* để phân biệt với *tệp* được dùng để chỉ một tập lớn các bản ghi được lưu trữ ở bộ nhớ ngoài (sắp xếp tương ứng với tệp sẽ gọi là *sắp xếp ngoài*).

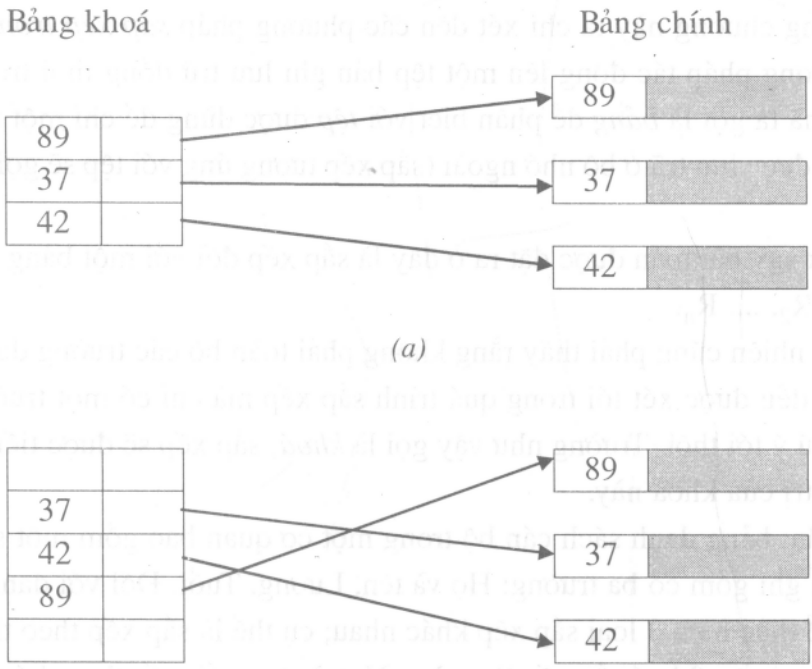
Như vậy bài toán được đặt ra ở đây là sắp xếp đối với một bảng gồm n bản ghi R_1, R_2, ..., R_n.

Tuy nhiên cũng phải thấy rằng không phải toàn bộ các trường dữ liệu trong bản ghi đều được xét tới trong quá trình sắp xếp mà chỉ có một trường nào đó được chú ý tới thôi. Trường như vậy gọi là *khoá*; sắp xếp sẽ được tiến hành dựa vào giá trị của khoá này.

Ví dụ: bảng danh sách cán bộ trong một cơ quan bao gồm một số bản ghi; mỗi bản ghi gồm có ba trường: Họ và tên, Lương, Tuổi. Đối với danh sách này ta có thể thực hiện 3 loại sắp xếp khác nhau; cụ thể là sắp xếp theo thứ tự A, B, C của họ và tên (khoá sắp xếp là trường Họ và tên), sắp xếp theo thứ tự tăng dần (hay giảm dần) của tuổi (khoá sắp xếp sẽ là trường Tuổi), sắp xếp theo thứ tự tăng dần (hay giảm dần) của lương (khoá sắp xếp sẽ là trường Lương).

Khi sắp xếp, các bản ghi trong bảng sẽ được bố trí lại các vị trí sao cho giá trị khoá sắp xếp tương ứng với chúng đúng như thứ tự ấn định. Ta thấy kích thước của khoá thường rất nhỏ so với kích thước của cả bản ghi. Nếu sắp xếp thực hiện trực tiếp trên các bản ghi của bảng sẽ kéo theo sự chuyển đổi vị trí của các bản ghi, điều này có nghĩa là phải sao chép lại toàn bộ thông tin của các bản ghi vào vị trí mới, gây tốn phí về thời gian.

Người ta khắc phục bằng cách xây dựng một bảng phụ, cũng bao gồm n bản ghi như bảng chính nhưng chỉ bao gồm hai trường: *khoá* và *con trỏ*. Trường *khoá* chứa giá trị của khoá ứng với từng bản ghi trong bảng chính, trường *con trỏ* chứa con trỏ chỉ tới bản ghi tương ứng; bảng phụ này gọi là *bảng khoá*, sắp xếp sẽ được thực hiện trực tiếp trên bảng đó. Như vậy, trong quá trình sắp xếp bảng chính không hề bị ảnh hưởng; còn việc truy nhập vào bản ghi nào đó của bảng chính, nếu cần thiết, vẫn có thể dựa vào con trỏ bản ghi tương ứng thuộc bảng khoá này.



Hình 7.1

a) Trước khi sắp xếp; b) Sau khi sắp xếp.

Vì khoá có vai trò quan trọng như vậy nên sau này, khi trình bày các giải thuật, các thí dụ minh hoạ ta có thể coi khoá như đại diện của các bản ghi và vì vậy ta chỉ nói tới giá trị của khoá.

Không làm giảm tính tổng quát của vấn đề ta hoàn toàn có thể giả sử giá trị của khoá là các số nguyên; các giải thuật, các mô tả, các thí dụ có thể thay đổi dễ dàng để xử lý các bản ghi.

2. Một số phương pháp sắp xếp

2.1. Sắp xếp kiểu lựa chọn (selection short)

Ý tưởng cơ bản của cách sắp xếp này là duyệt qua nhiều lần danh sách hay một phần của nó, và cứ sau mỗi lần duyệt có một phần tử được sắp xếp vào đúng vị trí. Ví dụ mỗi lần duyệt danh sách con, phần tử nhỏ nhất của nó được tìm ra và được chuyển đến vị trí đúng của nó.

Để minh hoạ, ta giả sử rằng cần sắp xếp danh sách sau đây theo thứ tự tăng dần: 67 33 21 84 49 50 75

Ta duyệt danh sách để tìm phần tử nhỏ nhất và thấy nó ở vị trí thứ 3:

67 33 **21** 84 49 50 75

Ta hoán vị phần tử này với phần tử đầu tiên và như vậy xác định đúng vị trí của phần tử nhỏ nhất là ở đầu danh sách:

21 33 67 84 49 50 75

Bây giờ ta duyệt con bao gồm các phần tử từ vị trí thứ 2 để tìm phần tử nhỏ nhất:

21 **33** 67 84 49 50 75

Và hoán vị nó với phần tử thứ 2, như vậy ta đã xác định đúng vị trí của phần tử nhỏ nhất tiếp theo ở vị trí thứ 2.

Cứ như vậy cho đến khi ta thực hiện quy trình trên với danh sách chỉ còn hai phần tử cuối cùng:

21 33 **49** 84 67 50 75
21 33 49 **50** 67 84 75
21 33 49 50 **67** 84 75
21 33 49 50 67 **75** 84

Nói tóm lại nguyên tắc như sau:

Ở lượt thứ i (i=1, 2, ..., n) sẽ chọn trong dãy giá trị k_i, k_{i+1}, ..., k_n giá trị nhỏ nhất và đổi chỗ cho k_i. Sau n-1 lượt sẽ hoàn thành việc sắp xếp.

Procedure selection_short(k,n);

{Cho dãy khoá k gồm n phần tử, giải thuật này thực hiện sắp xếp các phần tử của k theo thứ tự tăng dần, dựa vào phép chọn phần tử nhỏ nhất trong mỗi lượt duyệt}

```
1. For i:=1 to n-1 do
2.    Begin
          m:=i;
3.        For j:=i+1 to n do
              If  k[j] < k[m] then m:=j;
4.                If m<>i then
                                begin {đổi chỗ}
                                    X:=k[i];
                                    k[i]:=k[m];
                                    k[m]:=X;
                                end
```

End;

5. **Return**

Độ phức tạp tính toán:

Trong lần đầu tiên khi duyệt qua danh sách, phần tử đầu tiên được so sánh với n-1 phần tử còn lại, lần duyệt thứ 2 có n-2 phép so sánh,... như vậy có tất cả:

(n-1) + (n-2) + (n-3) + ... + 2 +1 = n(n-1)/2 lần so sánh, nghĩa là độ phức tạp tính toán của phương pháp này là: $O(n^2)$.

2.2. Sắp xếp kiểu chèn dần (thêm dần)

Nguyên tắc sắp xếp ở đây là dựa theo kinh nghiệm của người chơi bài: khi có i-1 lá bài đã sắp xếp trên tay, lấy thêm quân bài thứ i nữa thì so sánh với i-1, i-2,... để tìm chỗ thích hợp để chèn.

Ta có thể diễn đạt "thô" như sau: thoạt đầu k_1 được coi như một dãy chỉ bao gồm một phần tử đã được sắp thứ tự; xét thêm k_2, ta so sánh nó với k_1 để xác định chỗ chèn,... *Ở lượt thứ i, k_i sẽ được chèn vào đúng vị trí của nó trong dãy $k_1, k_2,..., k_{i-1}$ đã được sắp thứ tự; điều này ta thực hiện bằng cách so sánh k_i với từng phần tử $k_1, k_2,..., k_{i-1}$, bắt đầu từ bên phải và dịch chúng sang phải khi cần thiết.*

Vị trí 0 của mảng được giả sử bị chiếm bởi một giá trị nhỏ hơn mọi phần tử trong mảng để khỏi *rơi sang trái* trong những lần duyệt từ phải sang trái.

Trước hết ta minh hoạ giải thuật này với dãy:

67 33 21 84 49 50 75

Danh sách con có thứ tự tạo ra sau mỗi lượt được đánh dấu:

67	33	21	84	49	50	75	Danh sách con có thứ tự gồm một phần tử
33	67	21	84	49	50	75	Chèn 33 để có danh sách gồm 2 phần tử
21	33	67	84	49	50	75	Chèn 21 để có danh sách gồm 3 phần tử
21	33	67	84	49	50	75	
21	33	49	67	84	50	75	
21	33	49	50	67	84	75	
21	33	49	50	67	75	84	

Độ phức tạp tính toán:

Trường hợp xấu nhất của giải thuật này là các phần tử của dãy được sắp xếp hoàn toàn ngược lại; khi đó việc chèn k_2 đòi hỏi 2 lần so sánh (với k_1 rồi k_0), chèn k_3 cần 3 phép so sánh,... tổng số lần so sánh là:

$$2 + 3 + 4 + ... + n = n(n+1)/2 - 1$$

Vậy $T(n) = O(n_2)$;

Giải thuật sắp xếp kiểu chèn dần như sau:

Procedure Insert_short(k,n);

{Trong thủ tục này, người ta dùng X để chứa giá trị mới cần chèn, -∞ là một giá trị nào đó nhỏ hơn tất cả các giá trị trong danh sách}

1. k[0]:= - ∞;
2. **For** i:=2 **to** n **do Begin**
3. X:=k[i]; j:=i-1;
4. {Xác định chỗ giá trị mới được xét và dịch chuyển các giá trị khi cần thiết}
   While X < k[j] **do begin**
   k[j+1]:=k[j];
   j:=j - 1;
   end;
5. {Đưa X vào đúng vị trí}
   k[i+1]:=X;
   End;
6. **Return**.

2.3. Sắp xếp theo kiểu nổi bọt

Nguyên tắc: dãy giá trị được duyệt từ đáy lên đỉnh. Dọc đường gặp hai khoá kế cận ngược thứ tự thì đổi chỗ chúng cho nhau.

Giải thuật nổi bọt bắt đầu từ cuối bảng và tiến dần về đầu bảng; nó so sánh từng phần tử với từng phần tử đứng trước. Nếu hai phần tử không đúng thứ tự thì chúng sẽ đổi chỗ. Ở vòng đầu tiên phần tử nhỏ nhất sẽ "nổi" lên đỉnh; sau vòng 2, phần tử nhỏ thứ 2 sẽ được xếp vào vị trí thứ hai,...

Ta minh hoạ giải thuật này bằng thí dụ sau:

67 33 21 84 49 50 75

Bằng giải thuật nổi bọt, thứ tự cách phần tử sau từng vòng lặp sẽ như sau:

21 67 33 49 84 50 75
21 33 67 49 50 84 75
21 33 49 67 50 75 84
21 33 49 50 67 75 84
21 33 49 50 67 75 84
21 33 49 50 67 75 84

Sau đây là giải thuật:

Procedure BUBBLE_SHORT(k,n);

1. **For** i:=1 **to** n-1 **do**
2. **For** j:=n **downto** i+1 **do**
3. **If** k[j] < k[j-1] **then**

```
Begin
    x:=k[j];
    k[j]:= k[j-1];
    k[j-1]:=x;
End;
```

4. **Return**

3. Thời gian thực hiện giải thuật

3.1. Sắp xếp lựa chọn

Ở lượt thứ i (i-1, 2,... n-1) để tìm khoá nhỏ nhất cần Ci = (n-i) phép so sánh. Số lượng ghép so sánh không phụ thuộc vào dãy khoá ban đầu. Từ đó suy ra:

$$C_{\min} = C_{\max} = C_{tb} = \sum_{i=1}^{n-1}(n-1) = \frac{n(n-1)}{2}$$

3.2. Sắp xếp chèn dần

Số lượng ghép so sánh phụ thuộc vào khoá ban đầu. Trường hợp thuận lợi nhất ứng với dãy khoá đã được sắp xếp rồi. Như vậy mỗi lượt chỉ cần một phép so sánh. Do đó:

$$C_{\min} = \sum_{i=2}^{n} 1 = n-1$$

Nhưng nếu dãy khoá ban đầu có thứ tự ngược với thứ tự sắp xếp thì ở lượt thứ i phải cần có $C_i = (i-1)$ phép so sánh. Vì vậy:

$$C_{\max} = \sum_{i=2}^{n}(i-1) = \frac{n(n-1)}{2}$$

Nếu giả sử mọi giá trị khóa đều xuất hiện đồng khả năng thì trung bình ở lượt thứ i có thể coi như $C_i = \frac{i}{2}$ phép so sánh. Suy ra:

$$C_{tb} = \sum_{i=2}^{n}\frac{i}{2} = \frac{n^2+n-2}{4}$$

3.3. Sắp xếp nổi bọt

Với sắp xếp kiểu nổi bọt thì tương tự như phương pháp đầu, ta cũng có:

$$C_{\min} = C_{\max} = C_{tb} = \sum_{i=1}^{n-1}(n-1) = \frac{n(n-1)}{2}$$

Nhìn vào kết quả đánh giá ở trên ta thấy phương pháp sắp xếp kiểu chèn dần tỏ ra "tốt hơn" so với hai phương pháp kia. Tuy nhiên, với n khá lớn, chi phí về thời gian thực hiện được đánh giá qua cấp độ lớn thì cả ba phương pháp đều có cấp $O(n^2)$ và đây vẫn là một chi phí cao so với một số phương pháp mà ta sẽ xét thêm sau đây:

3.4. Sắp xếp kiểu phân đoạn (Partition_short)/ sắp xếp nhanh (Quick_short)

Sắp xếp kiểu phân đoạn là một cải tiến của phương pháp sắp xếp kiểu đổi chỗ. Đây là một phương pháp khá tốt, do đó người sáng lập ra nó là Hoare đã mạnh dạn đặt cho nó một cái tên hấp dẫn là sắp xếp NHANH.

Tư tưởng của phương pháp này có thể tóm tắt như sau:

Chọn một khoá ngẫu nhiên nào đó làm chốt. Mọi phần tử nhỏ hơn *chốt* phải được sắp xếp vào vị trí ở trước chốt, mọi phần tử lớn hơn khoá *chốt* phải được sắp xếp vào vị trí sau khoá *chốt*. Muốn vậy, các phần tử trong dãy sẽ được so sánh với khoá *chốt* và sẽ đổi vị trí cho nhau, hoặc cho *chốt*, nếu nó lớn hơn *chốt* mà lại nằm trước *chốt* hoặc nhỏ hơn *chốt* mà lại nằm sau *chốt*. Khi việc đổi chỗ đã thực hiện xong thì dãy khoá lúc đó được phân làm hai đoạn: một đoạn gồm các khoá nhỏ hơn *chốt*, một đoạn gồm các khoá lớn hơn *chốt*, còn khoá *chốt* thì nằm giữa hai đoạn nói trên. Đó chính là vị trí thực của nó trong dãy khi đã được sắp xếp. Tới đây coi như kết thúc một lượt sắp xếp.

Ở các lượt tiếp theo cũng áp dụng kỹ thuật tương tự đối với mỗi phân đoạn còn lại. Lẽ tất nhiên chỉ có một phân đoạn được xử lý ngay sau đó, còn một phân đoạn phải để lúc khác, nghĩa là phải được ghi nhớ lại.

Quá trình xử lý một phân đoạn, ghi nhớ phân đoạn còn lại được thực hiện tiếp tục cho đến khi gặp một phân đoạn chỉ gồm có hai phần tử thì việc ghi nhớ không cần nữa. Lúc đó một phân đoạn mới sẽ được xác định và đối với phân đoạn này quá trình lặp lại tương tự. Sắp xếp sẽ kết thúc khi phân đoạn cuối cùng đã được xử lý xong.

Với phương pháp sắp xếp này người ta đã chứng minh được rằng cấp độ thực hiện giải thuật là:

$$T_{tb}(n) = O(n\log_2 n)$$

Như vậy rõ ràng là khi n khá lớn QUICK_SHORT đã tỏ ra có hiệu lực hơn hẳn 3 phương pháp đã nêu trên.

Sau đây là giải thuật:

Giải thuật này thực hiện sắp xếp cho bảng khoá k gồm n bản ghi theo thứ tự tăng dần của khoá. Ở đây sử dụng một bản ghi giả có khoá tương ứng là K[n+1] mà k[i] ≤ k[n+1] với 1 ≤ i ≤ n. LB và UB là chỉ số dưới và chỉ số trên (2 biên) của bảng được xử lý. Lúc đầu LB = 1 còn UB = n. Hai biến chỉ số i và j được dùng để lựa chọn khoá trong quá trình xử lý. KEY là biến chứa giá trị của khoá *chốt*. B là biến logic, khi B = FALSE thì bảng được phân thành 2 bảng con.

```
Procedure Quick_short(K, LB, UB);
1. {Khởi dầu}
B:=True
2. {Thực hiện sắp xếp}
   if LB < UB then
    Begin
          i:=LB; j=UB+1; KEY:=K[LB];
            While B do
                Begin
                   i:=i+1;
                   While K[j] < KEY do i:=i+1;
                   j:=j-1;
                   While K[j] > KEY do j:=j-1;
                   if  i < j  then  K[i] <--> K[j] {đổi chỗ giữa K[i] và K[j]}
                            Else  B:=FALSE;
                End;
          K[LB] <--> K[j];
          Call Quick_short(K, LB, j-1);
          Call Quick_short(K, j+1. UB);
    end;
 3. Return
```

3.5. Sắp xếp kiểu vun đống (Heap_short)

Sắp xếp kiểu phân đoạn đã cho ta thời gian thực hiện trung bình khá tốt. Nhưng trong trường hợp xấu nhất nó vẫn là $O(n^2)$.

Phương pháp sắp xếp mà ta sẽ xét sau đây đã đảm bảo được trong mọi trường hợp chi phí thời gian đều cùng là $O(n\log_2 n)$.

Với phương pháp sắp xếp này, bảng khoá sẽ có cấu trúc cây nhị phân hoàn chỉnh và được lưu trữ kế tiếp trong máy.

Phương pháp sắp xếp này được thực hiện như sau:

Sắp xếp vun đống được chia làm hai giai đoạn.

Giai đoạn 1: Giai đoạn tạo đống. Ở giai đoạn này cây nhị phân biểu diễn bảng khoá được biến đổi để trở thành một đống. *Đống* ở đây là một *cây nhị phân hoàn chỉnh có xu hướng lệch trái* mà mỗi nút được gán một giá trị khoá sao cho khoá ở nút cha bao giờ cũng lớn hơn khoá ở nút con nó. Cây nhị phân hoàn chỉnh này được lưu trữ kế tiếp trong máy.

Giai đoạn 2: Giai đoạn sắp sếp, nhiều lượt xử lý được thực hiện, mỗi lượt ứng với hai phép:

- Đưa khoá trội về vị trí thực của nó bằng cách đổi chỗ cho khoá hiện đang ở vị trí đó. Nghĩa là đổi chỗ khoá ở đỉnh đống (nút gốc cây) với khoá ở cuối đống (nút lá cuối cùng ở vị trí trái nhất).
- Vun lại thành đống đối với cây gồm các khóa còn lại sau khi đã loại khoá trội ra ngoài.

Một lượt tiếp theo sẽ được thực hiện tương tự như vậy và cứ như thế cho tới khi mọi khoá đã vào đúng chỗ sắp xếp của nó.

Giải thuật như sau:

Giải thuật sắp xếp vun đống được viết thành hai thủ tục: thủ tục **Vundong** thực hiện việc chỉnh lý một cây nhị phân với gốc i để nó thoả mãn điều kiện của đống và thủ tục **HEAP_SHORT** thực hiện việc sắp xếp.

```
Procedure Vundong(i,n);
1.{Khởi đầu}
   Key:=K[i];  j:=2*i;
2. {Chọn con ứng với khoá lớn nhất trong hai con của i}
   While j ≤ n do Begin
         if  (j < n) and (K[j] < K[j+1]) then j:=j+1;
```

```
3.{So sánh khoá cha với khoá con lớn nhất}
        if  Key < K[j] then
            begin K[j div 2]:= Key; return; end;
        K[j div 2]:=K[j]; j:=j*2;
            End;
4. {Đưa Key vào vị trí của nó}
K[j div 2]:= Key;
5. Return

Procedure HEAP_SHORT(K,n);
1.{Tạo đống ban đầu}
   for i:=n div 2 downto 1 do Vundong(1,n);
2.{Sắp xếp}
   for i:= n-1 downto 1 do
        Begin
                X:=K[i]; K[1]:= K[i+1];
                K[i+1]:=X;
                Call Vundong(1,i);
        End;
3. Return
```

3.6. Sắp xếp kiểu hoà nhập (Merge sort)

Sắp xếp kiểu hoà nhập là phép hợp nhất các bản ghi của hai bảng đã được sắp xếp để ghép thành một bảng, có kích thước lớn hơn cũng được sắp xếp. Nguyên tắc thực hiện của nó khá đơn giản: so sánh hai khoá nhỏ nhất (hoặc lớn nhất) của hai bảng con, chọn khoá nhỏ hơn (hoặc lớn hơn) để đưa ra miền sắp xếp (một miền nhớ phụ có kích thước bằng tổng kích thước của hai bảng con) và đặt nó vào vị trí thích hợp. Khoá được chọn từ đấy bị loại ra khỏi bảng chứa nó. Quá trình như vậy cứ tiếp tục cho tới khi một trong hai bảng con đã cạn. Lúc đó chỉ cần chuyển toàn bộ phần đuôi của bảng còn lại ra miền sắp xếp là xong.

Ví dụ: Với hai bảng đã sắp xếp:

17, 42, 50, 76

08, 34, 85

Quá trình hoà nhập sẽ như sau:

Lượt 1: 17, 42, 50, 76

08 <---- 34, 85

Lượt 2: 42, 50, 76

08, 17 <---- . 34, 85

Lượt 3: 42, 50, 76

08, 17, 34 <---- 85

.....

Lượt 6:

08, 17, 34, 42, 50, 76 <---- 85

Lượt 7:

08, 17, 34, 42, 50, 76, 85

Sau đây là giải thuật hoà nhập hai đường

Procedure Merge_short(X, b, m, n, Z)

{Thủ tục này thực hiện hoà nhập hai bảng con đã được sắp xếp thành một bảng mới được sắp xếp}

```
1. i:= k:= b; j:= m+1;
2. while i ≤ m and j ≤ n do
        if X[i] ≤ X[j] then begin
                        Z[k] := X[i];
                        i:= i+1;
                              end
        else      begin
                Z[k]:= X[j];
                j:= j+1;
          end;
   k:= k+1;
3.{Một trong hai bảng con đã cạn trước}
   if i > m then (Zk,.....,Zn):= (Xj,... ,Xn)
            else (Zk,.....,Zn):= (Xi,... ,Xm)
4. Return
```

Vấn đề đặt ra trong thực tế là việc xắp xếp phải được tiến hành trên một bảng khoá chưa được sắp xếp. Vì vậy phương pháp sắp xếp kiểu hoà nhập tỏ ra bị hạn chế. Tuy nhiên từ phép hoà nhập hai đường người ta đã triển khai thành

một số phương pháp sắp xếp mới xuất phát từ nhận xét như sau: mỗi bản ghi có thể coi là một mạch đã được sắp xếp có độ dài bằng 1. Nếu hoà nhập hai bảng như vậy sẽ được một mạch mới có độ dài bằng 2. Lại hoà nhập hai mạch có độ dài bằng 2, ta được một mạch có độ.dài 4... Và cứ như thế, cuối cùng ta sẽ được một mạch có độ dài n, tức là bảng đã được sắp xếp hoàn toàn. Phương pháp sắp xếp như vậy được gọi là *sắp xếp hoà nhập hai đường tự nhiên.*

II. TÌM KIẾM

1. Khái niệm

Tìm kiếm là một nhu cầu rất thường xuyên trong đời sống hàng ngày cũng như trong xử lý tin học. Trong các phần trước nói riêng và trong các bộ môn khác nói chung chúng ta đã thấy xuất hiện những yêu cầu về tìm kiếm; ở những chừng mực nhất định chúng ta đã phần nào giải quyết vấn đề này. Trong chương này, bài toán tìm kiếm sẽ được xem xét một cách độc lập, toàn diện; chúng ta có thể trình bày bài toán như sau:

"Cho một bảng (dãy, danh sách) bao gồm n bản ghi R_1, R_2, ..., R_n. Mỗi bản ghi R_i tương ứng với một khoá k_i, hãy tìm bản ghi có khoá bằng một giá trị cho trước X". X được gọi là *khoá tìm kiếm hay đối trị tìm kiếm.*

Công việc tìm kiếm sẽ kết thúc khi một trong hai tình huống sau xảy ra:

1. Tìm được một bản ghi có khoá bằng X, lúc đó ta nói phép tìm kiếm *thành công* (thoả).

2. Không tìm được bản ghi nào có khoá bằng X, lúc đó ta nói phép tìm kiếm *không thành công* (không thoả).

Như vậy trong thực tế, các phần tử của danh sách là các bản ghi, và việc tìm kiếm là dựa trên cơ sở của một trường (khoá) nào đó; nghĩa là chúng ta tìm trong danh sách một bản ghi R_i chứa một giá trị cho trước của trường khoá của nó. Tuy nhiên để tiện khi trình bày các giải thuật ta giả sử rằng các R_i là những mục dữ liệu đơn giản (chỉ gồm một trường).

Sau đây ta xét một số phương pháp tìm kiếm.

2. Một số phương pháp tìm kiếm

2.1. Tìm kiếm tuần tự

Đây là kỹ thuật tìm kiếm đơn giản và cổ điển; nội dung của nó có thể diễn đạt như sau:

"Bắt đầu từ phần tử đầu tiên, lần lượt so sánh khoá tìm kiếm với khoá tương ứng của các bản ghi trong bảng, cho tới khi tìm được bản ghi mong muốn hoặc đã hết bảng mà chưa thấy".

Giải thuật như sau:

Function Tuantu(k,n,X)

{Cho dãy khoá k gồm n phần tử; xác định xem trong đó có phần tử nào bằng X hay không; nếu có trả về chỉ số của phần tử đó, nếu không trả về 0; trong hàm này có sử dụng một phần tử phụ k_{n+1} mà giá trị của nó chính là X}

1.{*Khởi đầu*}

i:=1; k[n+1] :=X;

2.{*Tìm phần tử trong dãy*}

while k[i] <> X **do** i:= i+1;

3.{*Tìm thấy hay không*}

if i = n+1 **then return**(0)

else return(i)

2.2. Tìm kiếm nhị phân

Tìm kiếm nhị phân là một phương pháp tìm kiếm khá thông dụng. Nó tương tự như khi ta tìm số điện thoại của một cơ quan trong bảng danh mục điện thoại hay khi ta tìm một từ trong từ điển. Tìm kiếm nhị phân được áp dụng trong trường hợp một dãy khoá đã có thứ tự; nội dung của giải thuật này là so sánh giá trị cho trước X với giá trị khóa ở "giữa dãy" khoá đang xét. Tìm kiếm sẽ kết thúc nếu X= k_i. Nếu X < k_i tìm kiếm lại được làm với k_1,..., k_{i-1}; nếu X > k_i tìm kiếm lại được làm với k_{i+1},..., k_n. Với dãy khóa sau, một kỹ thuật tương tự lại được sử dụng. Quá trình tìm kiếm được tiếp tục khi tìm thấy khoá mong muốn hoặc dãy khóa đã xét trở nên rỗng (không tìm thấy).

Ta có thể minh hoạ giải thuật này như sau:

Function Binary_search(k,n,X)

{Cho dãy k gồm n phần tử được sắp thứ tự; xác định xem trong đó có phần tử nào bằng X hay không; nếu có trả về chỉ số của phần tử đó, nếu không trả về 0}

1.{*Khởi đầu*}

l := 1; r := n;

2.{*Tìm*}

while l <= r **do begin**

```
3.{Tính chỉ số giữa}
        m :=(l +r) div 2;
4.{So sánh}
    if X < k[m] then  r := m -1
                else
                        if  X >  k[m] then  l:= m+1
                            else return(m)
                                    end;
5.{Không thành công}
    return(0)
```

Giải thuật này có thể viết dưới dạng đệ quy như sau:

Function RBinary_search(l, r, k, X)

{l, r là chỉ số dưới và chỉ số trên của dãy k, m chỉ số giữa, LOC là chỉ số ứng với khóa cần tìm; nếu không thành công LOC có giá trị 0}

```
1. if  l > r then LOC:=0;
          else  m := (l+r) div 2;
   if  X < k[m]  then LOC := RBinary_search(l,m,k,X)
          else  if  X < k[m] then LOC := RBinary_search(m+1,r,k,X)
                    else LOC := m;
2. Return (LOC)
```

Với giải thuật này người ta chứng minh được $T_{tb}(n) = O(\log_2 n)$.

Rõ ràng là so với tìm kiếm tuần tự, chi phí tìm kiếm nhị phân ít hơn khá nhiều. Tuy nhiên ta cũng thấy rằng trước khi sử dụng tìm kiếm nhị phân dãy khoá đã phải được sắp xếp rồi, nghĩa là thời gian sắp xếp cũng phải kể đến. Nếu dãy khoá luôn biến động thì lúc đó chi phí cho sắp xếp là khá tốn kém và đó chính là nhược điểm của phương pháp này.

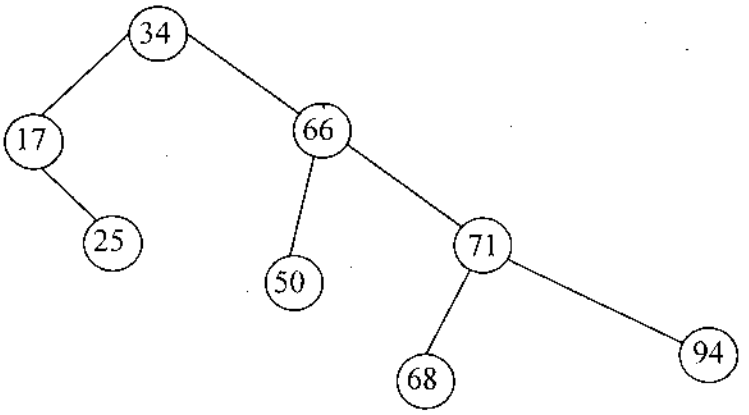
2.3. Cây nhị phân tìm kiếm (Binary Search Tree)

Để khắc phục nhược điểm vừa nêu trên của tìm kiếm nhị phân và đáp ứng yêu cầu với dãy khoá biến động, một phương pháp mới được đưa ra dựa trên cơ sở dãy được tổ chức dưới dạng cây nhị phân, mà ta gọi là *cây nhị phân tìm kiếm.*

2.3.1. Định nghĩa cây nhị phân tìm kiếm

Cây nhị phân tìm kiếm là một cây nhị phân mà khoá của nút trái nhỏ hơn khoá của nút gốc, khoá của nút gốc nhỏ hơn khoá của nút phải.

Ví dụ:



Hình 7.2

2.3.2. Giải thuật tìm kiếm

Để tìm kiếm một giá trị X nào đó trên một cây nhị phân tìm kiếm ta có thể thực hiện như sau:

So sánh X với khoá gốc và một trong 4 tình huống sau đây có thể xảy ra:

1) Không có gốc (cây rỗng): X không có trên cây, phép tìm kiếm không thoả.

2) X trùng với khoá ở gốc, phép tìm kiếm thoả.

3) X nhỏ hơn khoá ở gốc: tìm kiếm được tiếp tục bằng cách xét cây con trái của gốc với kỹ thuật tương tự.

4) X lớn hơn khoá ở gốc: tìm kiếm được tiếp tục bằng cách xét cây con phải của gốc với kỹ thuật tương tự.

Giải thuật có thể diễn đạt như sau:

Function BST(T,X)

{Thủ tục này thực hiện việc tìm kiếm trên cây nhị phân tìm kiếm, có gốc được trỏ bởi T, giá trị khoá cần tìm là X; nếu tìm kiếm được thoả thì đưa con trỏ q trỏ vào nút đó; nếu tìm kiếm không thoả thì bổ sung nút mới có khoá là X vào T và đưa con trỏ q trỏ vào nút mới đó}

1. *{Khởi tạo con trỏ}*

   p:=null; q:=T;

2. *{Tìm kiếm}*

   while q<> null **do**

   case

```
                X < key(q): p := q; q:= LPTR(q);
                X = key(q): return(q);
                X > KEY(q):  p:= q;  q:= RPTR(q);
            end case;
3.{Bổ sung}
    q <---- LAYTRA;
    KEY(q) := X;
    LPTR(q) := RPTR(q) := null;
    case
        T := null: T:= q; {cây rỗng đã bổ sung}
        X < KEY(p): LPTR(p):= q;
        else: RPTR(p):= q
    end case;
    write('Không thấy, đã bổ sung');
    return(q);
```

Với giải thuật trên có thể suy ra: Ta có thể dựng cây nhị phân tìm kiếm ứng với một dãy khoá đưa vào bằng cách liên tục bổ sung các nút ứng với từng khoá, bắt đầu từ một cây rỗng. Tất nhiên ban đầu phải dựng lên nút gốc cây ứng với khoá đầu tiên (trường hợp tìm kiếm trên cây rỗng) sau đó đối với các khoá tiếp theo, tìm trên cây không thấy thì bổ sung vào.

III. MỘT SỐ CHƯƠNG TRÌNH MINH HOẠ

1. Chương trình sắp xếp kiểu lựa chọn

```
PROGRAM Selection_sort;
uses crt;
const
   max=100;
type
   mang=array [1..max]  of  integer;
var
   day_so:mang;
   n,i,j,k:integer;
   tg:integer;
BEGIN
   clrscr;
```

```
    write('Nhap so phan tu day so n= ');readln(n);
    writeln;
    writeln('Nhap cac phan tu cua day:');
    for i:=1 to n do
      begin
          write('phan tu thu ',i,' = ');readln(day_so[i]);
      end;
    clrscr;
    writeln('Day so truoc khi sap xep:');writeln;
    for i:=1 to n do
        write(day_so [i]:4);writeln;
    {doan chuong trinh sap xep}
    for i:=1 to n-1 do
      Begin
          j:=i;
          for k:=j+1 to n do
              if day_so [k]<day_so [j]  then j:=k;
          if j<>i then Begin
                          tg:=day_so[i];
                          day_so [i]:=day_so [j];
                          day_so [j]:=tg;
                       End;
       End;
    writeln('Day so sau khi sap xep:');writeln;
    for i:=1 to n do
         write(day_so [i]:4);writeln;
    readln;
END.
```

2. Chương trình sắp xếp kiểu chèn dần

```
Program Insertion_sort;
  uses crt;
  const
       max=99;
       n=20;
  var
```

```
      x,i,j:integer;
      a:array [1..20] of integer;
Begin
   for i:=1 to n do {tao day ban dau ngau nhien}
       a [i] :=random(max);
   clrscr;
   writeln('day so ban dau');
   for i:=1 to n do write(a [i] :3);writeln;
   {doan chuong trinh sap xep}
   for i:= 2 to n do
      begin
          x:=a [i];
          j:=i-1;
          while x<a [j] do
             begin
                a [j+1] :=a [j];
                j:=j-1;
             end;
          a [j+1] :=x;
      end;
   writeln('day so da sap xep');
   for i:=1 to n do
   write(a [i] :3);
  readln;
End.
```

3. Chương trình sắp xếp kiểu đổi chỗ (nổi bọt)

```
Program noi_bot;
   uses crt;
   const
        max=99;
        n=20;
   var
        a:array [1..n] of integer;
        i,j,tg:integer;
   Begin
       for i:=1 to n do {tao day ban dau ngau nhien}
```

```
        a [i] :=random(max);
    clrscr;
    writeln('day so ban dau');
    for i:=1 to n do write(a [i] :3);writeln;
    {doan chuong trinh sap xep}
    for i:=1 to n-1 do
        for j:=n downto i+1 do
            if a [j] < a [j-1] then
                begin
                    tg:=a [j];
                    a [j] :=a [j-1];
                    a [j-1] :=tg;
                end;
    writeln('day sau khi sap xep');
    for i:=1 to n do
        write(a [i] :3);
    readln;
End.
```

4. Chương trình sắp xếp kiểu phân đoạn (sắp xếp nhanh)

```
Program Quick_sort;
    uses crt;
    const
        max=99;
        n=20;
    var
        pt:array[1..n] of integer;
        m,i,j,t,tg:integer;
(*-----------------*)
procedure sort(l,r:integer);
  Begin
    i:=l;
    j:=r;
    t:=pt [i];
    repeat
        while pt [i]<t do i:=i+1;
```

```
        while pt [j]>t do j:=j-1;
        if i<=j then
            begin
                tg:=pt [i];
                pt [i] :=pt [j];
                pt [j] :=tg;
                i:=i+1;j:=j-1;
            end;
    until i>j;
    if l<j then sort(l,j);
    if i<r then sort(i,r);
  end;
(*-------------------*)
BEGIN
    clrscr;
    m:=n;
    for i:=1 to m do
       pt [i] :=random(max);
    writeln('Day ban dau');
    for i:=1 to m do
       write(pt [i] :3);writeln;
    sort(1,m); {goi thu tuc sap xep}
    writeln('Day sau khi sap xep');
    for i:=1 to m do
    write(pt [i] :3);
    readln;
END.
```

5. Chương trình sắp xếp kiểu hoà nhập (Merge sort)

```
{Tron 2 mang da sap xep thanh 1 mang moi duoc sap xep
 theo thuat toan sap xep hoa nhap}
Program Merge_sort;
  Uses crt;
  Const
      max=50;
  Type
```

```
     mang = array [1..max] of integer;
Var
     A,B,C:mang;
     n,m:integer;
(*--------------*)
{thu tuc nhap du lieu}
Procedure Nhap(var mang_X:mang;n:integer;tenmang:char);
     Var
        i:integer;
     Begin
        writeln;
        for i:=1 to n do
           Begin
              write(tenmang,'[ ',i,'] = ');
              readln(mang_X[i]);
           End;
     End;
(*--------------*)
{ Thu tuc in du lieu}
Procedure List(Mang_X:mang;n:integer);
     Var
        i:integer;
     Begin
        writeln;
        for i:=1 to n do
           write(Mang_X[i]:4);
        writeln;
     End;
(*--------------*)
```

{Thu tuc sap xep theo thuat toan hoa nhap. A va B la 2 mang dua vao SX,mang A co m phan tu, mang B co n phan tu. Ket qua se dua vao mang C,so phan tu cua mang C la (m+n) phan tu}

```
Procedure SX_Hoanhap(A,B:mang;var C:mang;m,n:integer);
    Var
      i,j,k:integer;
    Begin
      if m=0 then  {Mang A khong co phan tu}
         Begin
            k:=1;
            for i:=1 to n do
               begin
                  C [k] :=B [i] ;k:=k+1;
               end;
         End;
      if n=0 then  {Mang B khong co phan tu}
         Begin
            k:=1;
            for i:=1 to m do
               begin
                  C [k] :=A [i] ;k:=k+1;
               end;
         End;
      {Sap xep khi A va B deu co phan tu}
      i:=1;j:=1;k:=1;
      While (i<=m) and (j<=n) do
         Begin
            If A [i] <=B [j] then
                  Begin
                     C [k] :=A [i] ;
                     i:=i+1;
                  End
             Else
                  Begin
                     C [k] :=B [j] ;
                     j:=j+1;
                  End;
```

```
            k:=k+1;
        End;
        If i>m then {Truong hop mang A het phan
                                        tu truoc}
            Begin
                while j<=n do
                    begin
                        C [k] :=B [j] ;
                        j:=j+1;
                        k:=k+1;
                    end
            End
         Else {Truong hop mang B het phan tu truoc}
            Begin
                while i<=m do
                    begin
                        C [k] :=A [i] ;
                        i:=i+1;
                        k:=k+1;
                    end
            End;
    End;
(*---------------*)
BEGIN
    clrscr;
    write('Nhap vao so phan tu cua mang A: ');
    readln(m);
    nhap(A,m,'A');
    write('Nhap vao so phan tu cua mang B: ');
    readln(n);
    nhap(B,n,'B');
    clrscr;
    Writeln('mang A'); List(A,m);
    Writeln('Mang B'); List(B,n);
    SX_hoanhap(A,B,C,m,n);
```

```
        Writeln('Mang C'); List(C,m+n);
        readln
End.
```

6. Chương trình sắp xếp kiểu hoà nhập hai đường tự nhiên

```
{Chuong trinh sap xep tep co cau truc bang phuong
phap tron tu nhien}
Program SXTepCTTN;
    uses crt;
    Type
        Pt=record
            key:integer;
            ten:string [30];
        end;
        tep=file of pt;
    Var
        A,B,C:tep;
        na,nb,nc,SoRun:integer;
        kt:boolean;
        a1,a2,b1,b2,c1,c2:pt;
(*----------------*)
{Ngoai tep goc can sap xep ky hieu la A, ta can tao
them hai tep phu la B va C co cung cau truc voi A}
    Procedure MoCacTep;
        var
            TenTep:string;
        Begin
            clrscr;
            write('Cho ten tep can sap xep');
            readln(TenTep);
            Assign(A,tentep);
            Assign(B,'c:\tp\bin\phu1.dat');
            Assign(C,'c:\tp\bin\phu2.dat');
        End;
(*----------------*)
{Tep can xem chinh la tep can sap xep va tep nay
```

```
da duoc mo}
        Procedure Xemtep;
            Begin
                  Reset(A);
                  while not EOF(A) do
                       begin
                            read(a,a1);
                            writeln(a1.key,'  ',a1.ten);
                       end;
            End;
   (*----------------*)
   {Copy 1 Run tu tep A sang tep D: Tep D co the la B
hoac C}
        Procedure CopyRun(var D:tep);
            Begin
                 Repeat
                      a1:=a2;write(D,a1);na:=na-1;
                      if na<=0 then Kt:=true
                               else begin
                                       read(A,a2);
                                       kt:=a1.key>a2.Key;
                                    end;
                 Until kt;
            End;
   (*----------------*)
   {Phan phoi cac Run tu A sang B va C}
        Procedure PhanPhoi;
            Begin

                 Reset(A);Rewrite(B);
                 Rewrite(C);na:=filesize(A);
                 Read(A,a2);
                 Repeat
                    CopyRun(B);
                    if na>0 then CopyRun(c);
                 Until na<=0;
```

```
            End;
    (*----------------*)
    {Copy 1 phan tu tu tep D vao tep A, m la so phan tu
hien tai cua D}
    Procedure CopyPt(var D:tep;var m:integer;var d1,d2:pt);
            Begin
                write(A,d1);m:= m-1;
                if m<=0 then kt:= true
                        else begin
                                Read(D,d2);
                                kt:=d1.key>d2.key;
                             end;
            End;
    (*----------------*)
    { Tron 2 Run bat ky tu B va C vao A}
        Procedure TronRun;
            Begin
                Repeat
                    b1:=b2;c1:=c2;
                    if b1.key<=c1.key then
                       CopyPt(B,nb,b1,b2)
                    else CopyPt(C,nc,c1,c2);
                Until kt;
                If b1.key<=b2.key then
                      repeat
                          b1:=b2; CopyPt(B,nb,b1,b2);
                      until kt;
                If c1.key<=c2.key then
                      repeat
                          c1:=c2; CopyPt(C,nc,c1,c2);
                      until kt;
            End;
    (*----------------*)
    { Noi phan con lai cua D (la B hoac C)vao A}
      Procedure NoiVaoA(var D:tep;var m:integer;p:pt);
```

```
        Begin
            SoRun:=SoRun+1;
            Repeat
                write(A,p);m:=m-1;
                if m>0 then read(D,p);
            Until m<=0;
        End;
(*----------------*)
    {Tron cac Run tu B va C vao A}
    Procedure tron;
        Begin
            ReWrite(A);Reset(B);Reset(C);
            nb:=Filesize(B);nc:=FileSize(C);
            SoRun:=0;
            if (nb>0) and (nc>0) then
                begin
                  Read(B,b2);Read(C,c2);
                  Repeat
                        TronRun;SoRun:=SoRun+1;
                  until (nb<=0) or (nc<=0);
                  if nb >0 then NoivaoA(B,nb,b2);
                  if nc >0 then NoivaoA(C,nc,c2);
                end
            else
                begin
                    if nb >0 then begin
                                    Read(B,b1);
                                    NoiVaoA(B,nb,b1);
                                  end;
                    if nc >0 then begin
                                    Read(C,c1);
                                    NoiVaoA(C,nc,c1);
                                  end;
                 end;
        End;
(*----------------*)
```

```
    Procedure SXTronTN;
        Begin
            Repeat
                PhanPhoi;
                Tron;
            Until SoRun=1;
        End;
(*--------------*)
    BEGIN
        MoCacTep;Writeln('Tep truoc khi sap xep');
        XemTep;
        SxTronTN;
        Writeln('Tep sau khi sap xep');XemTep;
        Readln;
    END.
```

7. Chương trình tìm kiếm có bổ sung trên cây nhị phân tìm kiếm theo giải thuật BST

```
Program BST;
   Uses crt;
   Type
        P=^cay;
        cay = record
                nutgoc:string [20];
                trai,phai:P;
              end;
   Var
      Nutmoi:string [20];
      goc:P;
(*--------------*)
   Procedure Xuli(Nut:P);
        begin
           write(Nut^.nutgoc,' ->');
        end;
(*--------------*)
   Procedure duyet_truoc(Nut:P);
```

```
        begin
            if Nut<>NIL then
                begin
                    Xuli(Nut);
                    duyet_truoc(Nut^.trai);
                    duyet_truoc(Nut^.phai);
                end;
        end;
(*---------------*)
    Procedure duyet_giua(Nut:P);
        begin
            if Nut<>NIL then
                begin
                    duyet_giua(Nut^.trai);
                    xuli(Nut);
                    duyet_giua(Nut^.phai);
                end;
        end;
(*---------------*)
    Procedure duyet_sau(Nut:P);
        begin
            if Nut<>NIL then
                begin
                    duyet_sau(Nut^.trai);
                    duyet_sau(Nut^.phai);
                    xuli(Nut);
                end;
        end;
(*---------------*)
    Procedure tim(var goc :P);
   {Tim kiem trong cay xem co du lieu trong cay chua.
     Neu chua thi chen them vao}
        begin
            if goc = NIL then
                begin
```

```
                    NEW(goc);
                        begin
                            goc^.nutgoc :=Nutmoi;
                            goc^.trai:=NIL;
                            goc^.phai:=NIL;
                        end;
                end
            else
                if goc <>nil then
                    with goc^ do
                        begin
                            if Nutmoi < nutgoc then
                                tim(goc^.trai)
                            else
                                if Nutmoi > nutgoc then
                                    tim(goc^.phai)
                                else
                                  writeln('Tim thay');
                        end;
        end;
(*---------------*)
    Procedure in_cay(var cnp:p;s:integer);
        var
            i:integer;
        Begin
            if cnp <> nil then
            with cnp^ do
                begin
                    in_cay(trai,s+2);
                    for i:=1 to s do
                        write('   ');
                    writeln(nutgoc);
                    in_cay(phai,s+2);
                end;
        End;
```

```
(*---------------*)
   BEGIN
      clrscr;
      writeln('Chuong trinh duyet cay nhi phan:');
      goc:=nil;
      repeat
           write('Nhap: gia tri nut la 0 de ket thuc nhap:');
           readln(Nutmoi);
           if length(Nutmoi)>0 then
                    tim(goc);
      until length(Nutmoi)= 0;
      writeln;clrscr;
      writeln('IN CAY NHI PHAN DA SAP XEP');
      in_cay(goc,0);
      readln;
      writeln('Duyet Truoc : ');
      duyet_truoc(goc);
      writeln;
      writeln('Duyet Giua : ');
      duyet_giua(goc);
      writeln;
      writeln('Duyet Sau : ');
      duyet_sau(goc);
      writeln;
      readln;
   END.
```

Bài tập

1. Cho dãy khoá

50 08 34 06 98 17 83 25 66 42 21 59 62 71 85 76

Hãy minh hoạ ba phương pháp sắp xếp dưới đây qua dãy khoá trên theo thứ tự tăng dần, thứ tự giảm dần:

- Sắp xếp lựa chọn
- Sắp xếp chèn dần
- Sắp xếp đổi chỗ (nổi bọt)

2. Hãy sửa lại ba giải thuật dưới đây để thực hiện sắp xếp theo thứ tự giảm dần.

- Sắp xếp lựa chọn
- Sắp xếp chèn dần
- Sắp xếp đổi chỗ (nổi bọt)

3. Với dãy khoá đã cho ở bài 1 hãy minh hoạ hai phương pháp sắp xếp phân đoạn và sắp xếp kiểu vun đống.

4. Nếu thực hiện sắp xếp theo thứ tự giảm dần thì *đống* sẽ được định nghĩa thế nào?

5. Với dãy khoá cho ở bài 1 hãy thực hiện sắp xếp kiểu hoà nhập hai đường tự nhiên.

6. Hãy dựng cây nhị phân tìm kiếm theo giải thuật BST đối với dãy khoá đã cho ở bài 1. Hãy đánh dấu đường đi trên cây này khi thực hiện tìm kiếm khoá 42 và 83.

TÀI LIỆU THAM KHẢO

1. ***Cấu trúc dữ liệu và giải thuật***, Đỗ Xuân Lôi, Nhà xuất bản Khoa học và Kỹ thuật.

2. ***Cấu trúc dữ liệu và giải thuật***, Đinh Mạnh Tường, Nhà xuất bản Khoa học và Kỹ thuật.

3. ***Thuật toán***, Nguyễn Xuân Huy, Nhà xuất bản Thống kê.

4. ***Giáo trình đồ thị***, Nguyễn Đức Nghĩa - Trường Đại học Bách khoa Hà Nội.

5. ***Toán rời rạc và ứng dụng trong tin học***, người dịch Phạm Thiều.

6. ***Lập trình nâng cao bằng Pascal với các cấu trúc dữ liệu***, Larry N. Hoff, Sanford Leestma, bản dịch của Lê Minh Trung.

7. ***Sắp xếp và tìm kiếm dữ liệu trên máy tính điện tử***, Đỗ Xuân Lôi - Đại học Bách khoa Hà Nội.

8. ***Toán rời rạc***, Nguyễn Đức Nghĩa, Nguyễn Tô Thành, Nhà xuất bản Giáo dục.

9. ***C++ và lập trình hướng đối tượng***, GS. Phạm Văn Ất, Nhà xuất bản Khoa học và Kỹ thuật.

10. ***Lập trình bằng C++***, PTS. Dương Tử Cường, Nhà xuất bản Khoa học và Kỹ thuật.

11. ***Bài tập Pascal***, Dương Viết Thắng chủ biên - Đại học Bách khoa Hà Nội.

12. ***Algorithms + data structures = programs***, Niklaus Wirth.

MỤC LỤC

NHÀ XUẤT BẢN HÀ NỘI
SỐ 4 - TỐNG DUY TÂN, QUẬN HOÀN KIẾM, HÀ NỘI
Điện thoại: (04)8.252916. Fax: (04)9.289143

GIÁO TRÌNH

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

NHÀ XUẤT BẢN HÀ NỘI - 2007

Chịu trách nhiệm xuất bản:
NGUYỄN KHẮC OÁNH

Biên tập:
NGUYỄN HUỲNH MAI
Bìa:
TRẦN QUANG
Kỹ thuật vi tính:
HOÀNG LAN HƯƠNG
Sửa bản in:
PHẠM TRANG

In 500 cuốn, khổ 17x24cm, tại Nhà in Hà Nội - Công ty Sách Hà Nội. 67 Phó Đức Chính - Ba Đình - Hà Nội. Quyết định xuất bản: 160-2007/CXB/443GT-27/HN, số 313/CXB ngày 02/3/2007. Số in: 390. In xong và nộp lưu chiểu quý III năm 2007.

BỘ GIÁO TRÌNH XUẤT BẢN NĂM 2007
KHỐI TRƯỜNG TRUNG HỌC CÔNG NGHIỆP

1. THỰC TẬP QUA BAN HÀN
2. THỰC TẬP QUA BAN NGUỘI
3. THỰC TẬP QUA BAN MÁY
4. AN TOÀN LAO ĐỘNG CHUYÊN NGÀNH SCKTTB
5. AN TOÀN LAO ĐỘNG CHUYÊN NGÀNH ĐIỆN
6. VẬT LIỆU ĐIỆN
7. ĐO LƯỜNG ĐIỆN
8. CƠ SỞ KỸ THUẬT ĐIỆN
9. ĐIỆN TỬ CÔNG SUẤT
10. MÁY CÔNG CỤ CẮT GỌT
11. ĐỒ GÁ
12. CÔNG NGHỆ CHẾ TẠO MÁY
13. TỔ CHỨC SẢN XUẤT
14. MÁY VÀ LẬP TRÌNH CNC
15. LÝ THUYẾT CHUYÊN MÔN TIỆN
16. SỬA CHỮA MÁY CÔNG CỤ
17. MÁY ĐIỆN
18. TRUYỀN ĐỘNG ĐIỆN
19. KHÍ CỤ ĐIỆN - TRANG BỊ ĐIỆN
20. CUNG CẤP ĐIỆN
21. KỸ THUẬT ĐIỀU KHIỂN LOGÍC VÀ ỨNG DỤNG
22. HƯỚNG DẪN ĐỒ ÁN CÔNG NGHỆ CTM
23. THỰC HÀNH CẮT GỌT KIM LOẠI
24. THỰC HÀNH SỬA CHỮA MÁY CÔNG CỤ
25. THÍ NGHIỆM KỸ THUẬT ĐIỆN
26. THÍ NGHIỆM MÁY ĐIỆN
27. THỰC TẬP ĐIỆN CƠ BẢN
28. TIẾNG ANH CHUYÊN NGÀNH SCKTTB
29. TIẾNG ANH CHUYÊN NGÀNH ĐIỆN
30. QUẢN TRỊ DOANH NGHIỆP
31. HƯỚNG DẪN ĐỒ ÁN TRANG BỊ ĐIỆN
32. HƯỚNG DẪN ĐỒ ÁN CUNG CẤP ĐIỆN
33. CƠ SỞ THIẾT KẾ MÁY
34. ĐỒ ÁN CƠ SỞ THIẾT KẾ MÁY (ĐỒ ÁN CHI TIẾT MÁY)
35. CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT
36. LÝ THUYẾT TRUYỀN TIN
37. CƠ SỞ KỸ THUẬT TRUYỀN SỐ LIỆU
38. ASSEMBLY
39. THỰC TẬP CHUYÊN NGÀNH ĐIỆN
40. THỰC HÀNH PLC
41. FOXPRO

Giá: 23.500đ