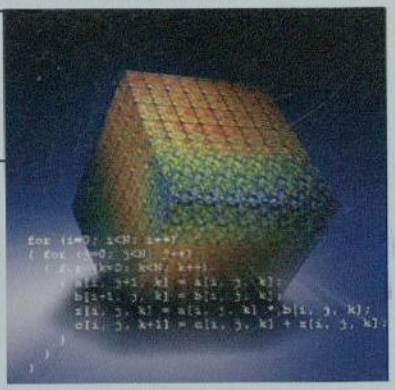
PGS. TS. HOÀNG NGHĨA TÝ



CẤU TRÚC DỮ LIỆU VÀ THUẬT TOÁN

NHÀ XUẤT BẢN XÂY DỰNG

PGS. TS. HOÀNG NGHĨA TÝ

CẤU TRÚC DỮ LIỆU
VÀ THUẬT TOÁN

NHÀ XUẤT BẢN XÂY DỰNG
HÀ NỘI - 2013

LỜI NÓI ĐẦU

Cấu trúc dữ liệu và thuật toán là môn học cơ sở ngành của ngành công nghệ thông tin. Chúng ta hãy hình dung một tòa nhà có phần nền móng, phần tường cột (kết cấu chịu lực) và những phần khác còn lại, nếu phần kết cấu chịu lực không vững thì tòa nhà không thể vững được. Môn "Cấu trúc dữ liệu và thuật toán" chính là môn học thuộc phần rường cột, phần kết cấu chịu lực đó của tòa nhà "Công nghệ thông tin". Khi học môn học này đòi hỏi sinh viên phải có đầu óc tư duy logic toán học để hiểu thuật toán, đồng thời phải có kỹ năng lập trình để diễn đạt sơ đồ thuật toán thành câu lệnh. Có thể sơ đồ thuật toán vẽ hết cả một trang giấy nhưng chuyển sang lập trình chỉ có vài dòng; có thể sơ đồ thuật toán trình bày theo kiểu kiểm tra điều kiện để tiếp tục ngay ở đầu nhưng khi chuyển sang lập trình có thể dùng các câu lệnh có cấu trúc điều khiển lặp với kiểu kiểm tra điều kiện trước hoặc sau...

Giáo trình ***"Cấu trúc dữ liệu và thuật toán"*** *này được xuất bản lần đầu năm 2006, do Nhà xuất bản Xây dựng ấn hành. Từ đó đến nay thời lượng cũng như kết cấu môn học đã có nhiều thay đổi. Ở Trường Đại học Xây dựng từ năm 2009 môn "Cấu trúc dữ liệu và thuật toán" đã được tách thành hai học phần là "Nhập môn Cấu trúc dữ liệu và thuật toán" và "Cấu trúc dữ liệu và thuật toán nâng cao" với mỗi học phần có 2 tín chỉ. Lý do là vì trong Khoa Công nghệ thông tin có nhiều chuyên ngành khác nhau, có chuyên ngành chỉ cần học một học phần, có chuyên ngành cần phải học cả hai học phần.*

Để đáp ứng sự thay đổi trên, trong mấy năm qua tác giả đã biên soạn lại đề cương chi tiết cho từng học phần và nội dung chương mục tương ứng, đã viết lại một số chương trình cho một số thuật toán, trình bày lại một số sơ đồ thuật toán (hoán vị ma trận thưa, danh sách liên kết, sắp xếp theo kiểu chèn..), viết thêm một số nội dung cho học phần Cấu trúc dữ liệu và thuật toán nâng cao (kiến thức nâng cao về con trỏ, đệ quy,

duyệt cây nhị phân, file mã, file text, danh sách liên kết, danh sách liên kết vòng và liên kết đôi...).

Với thời lượng 2 tín chỉ, nội dung học phần "Nhập môn Cấu trúc dữ liệu và thuật toán" sẽ được trình bày ở chương 1, chương 2, các mục 3.1, 3.2, 3.3 của chương 3, riêng các mục 3.3 đến 3.8 chỉ dừng ở các kiến thức cơ bản, chương 4, chương 5, chương 7 (mục 7.1, 7.2, 7.3), chương 8 (mục 8.1, 8.2).

Với thời lượng 2 tín chỉ, nội dung học phần "Cấu trúc dữ liệu và thuật toán nâng cao" sẽ ở phần mở rộng, nâng cao trong các mục 3.3 đến 3.8 của chương 3, chương 6, chương 7 (mục 7.4, 7.5), chương 8 (mục 8.3), chương 9.

Ngoài những phần lý thuyết và chương trình ví dụ, trong tài liệu còn giới thiệu một số chương trình tổng hợp, giúp cho bạn đọc tìm hiểu sâu hơn các thuật toán đã trình bày. Đây thực chất là một số kết quả của nhiều năm nghiên cứu và giảng dạy công nghệ thông tin cũng như hướng dẫn sinh viên của tác giả: chương trình quản lý tiền lương được soạn lại trên cơ sở phần mềm tính lương cho Phòng Tài vụ trường ĐHXD những năm 1990, chương trình sơ đồ mạng được viết từ những năm 1980-1981 chạy trên máy tính Minsk-32, chương trình quy hoạch động là diễn giải thuật toán của đề tài "Thiết kế tối ưu kết cấu mặt đường mềm nhiều lớp" được tiến hành một cách nung nấu kiên trì trong những năm 1975-1977, chương trình quản lý thi trắc nghiệm là một phần của đề tài "Xây dựng Ngân hàng đề thi và phần mềm phục vụ thi trắc nghiệm môn Tin học đại cương" năm 2001.

Quyển sách này được dùng làm giáo trình cho học viên, sinh viên ngành công nghệ thông tin và các ngành khác có học các môn tin học ứng dụng. Tác giả rất cảm ơn sinh viên và đồng nghiệp trong quá trình làm việc, giúp phát sinh ý tưởng hoàn thiện giáo trình, tuy đã rất cố gắng nhưng không thể tránh khỏi thiếu sót. Rất mong nhận được góp ý của độc giả để lần tái bản sau sách được hoàn thiện hơn. Mọi góp ý xin gửi về Bộ môn Công nghệ phần mềm, Khoa Công nghệ thông tin Trường đại học Xây dựng Hà Nội.

Tác giả

Phần I
CẤU TRÚC DỮ LIỆU

Chương 1
NHẬP MÔN CẤU TRÚC DỮ LIỆU

1.1. KHÁI NIỆM CẤU TRÚC DỮ LIỆU

Thuật toán và cấu trúc dữ liệu được coi là môn học cốt lõi của ngành Công nghệ thông tin. Niklaus Wirth - tác giả cuốn *"Cấu trúc dữ liệu + Giải thuật = Chương trình"* đã phân tích tầm quan trọng của cấu trúc dữ liệu và thuật toán. Chương trình là những mô tả cụ thể của các thuật toán trừu tượng dựa vào những biểu diễn và cấu trúc dữ liệu đặc biệt. Chương trình và dữ liệu không thể tách rời nhau được. Dữ liệu được xử lý bởi chương trình, cần được tổ chức thành các cấu trúc sao cho nó phản ánh mối quan hệ giữa các mục dữ liệu và cho phép xử lý dữ liệu có hiệu quả.

1.1.1. Dữ liệu ở dạng mã máy và ý nghĩa của chúng

Trong máy tính các giá trị dữ liệu được lưu trữ dưới dạng các bit, là các số ở hệ đếm nhị phân (0 và 1). Các bit được tổ chức thành các nhóm gọi là các Từ máy (word), tức là mỗi word chứa một số cố định các bit. Độ dài của word thay đổi theo loại máy tính (máy 16 bit hay 32 bit). Các *Từ máy* này được đánh địa chỉ bắt đầu từ 0, vì vậy có thể truy cập vào một word bất kỳ theo địa chỉ của nó.

Khi ở dạng một dãy bit, nó có thể biểu diễn nhiều ý nghĩa khác nhau.

Trong từng ngôn ngữ lập trình cụ thể, người lập trình sẽ biểu diễn các dãy bit này dưới dạng các cấu trúc xác định, cho phép xử lý và diễn đạt được các dữ liệu.

Những khái niệm quan trọng liên quan đến dữ liệu là Kiểu dữ liệu, Cấu trúc dữ liệu, Cấu trúc dữ liệu tĩnh, Cấu trúc dữ liệu động.

1.1.2. Kiểu dữ liệu

Các kiểu dữ liệu trong Pascal như integer, real, char, boolean, byte được gọi là kiểu dữ liệu đơn giản vì chúng là các dữ liệu không phân chia được nữa (gọi là các nguyên tử - atom). Tuy vậy chúng có thể được xem là có cấu trúc dữ liệu mà việc cài đặt chúng sử dụng các word như là cấu trúc lưu trữ cùng với các thuật toán để thực hiện những phép tính thích hợp cho từng kiểu.

Khái niệm *kiểu dữ liệu* rất quan trọng, nó xác định tập hợp giá trị mà biến có thể nhận. Mô tả kiểu sẽ quy định loại giá trị của biến và cung cấp cho chương trình dịch những thông tin cần thiết. Tương ứng với mỗi kiểu, mỗi dãy bit sẽ có một ý nghĩa riêng.

Kiểu dữ liệu còn quy định những phép toán có thể thực hiện được đối với các dữ liệu.

Mỗi ngôn ngữ lập trình định nghĩa một tập các kiểu dữ liệu nguyên thuỷ, vô hướng của riêng nó, ví dụ, trong ngôn ngữ Pascal đó là Integer, real, boolean, char; trong ngôn ngữ C đó là Int, float, char...

Kiểu dữ liệu có các đặc điểm:

+ Một kiểu dữ liệu sẽ xác định tập hợp giá trị mà một hằng trị (const) hay một biến sẽ nhận giá trị thuộc miền đó;

+ Kiểu của một giá trị được chỉ định bởi một hằng trị (const), biến hoặc biểu thức có thể suy ra từ dạng của nó hoặc từ khai báo mà không cần phải tiến hành quá trình tính toán.

+ Mỗi tác tử hay hàm có biến với kiểu định sẵn và cho ra kết quả cũng có kiểu định sẵn. Nếu một tác tử nhận một số biến có kiểu khác nhau (ví dụ phép cộng + dùng hai số nguyên hay thực) thì kiểu của kết quả được xác định theo quy tắc riêng của ngôn ngữ. Các kiểu integer, real, char, boolean, có thể biểu diễn bởi xâu các bit.

Sau đây ta xét một số kiểu dữ liệu điển hình trong Pascal.

a) Dữ liệu kiểu nguyên - integer

Dữ liệu kiểu integer được lưu trữ trong các *từ máy*. Như vậy độ dài của *từ máy* giới hạn phạm vi của các số nguyên lưu trữ được. Số nguyên lớn

nhất lưu trữ được trong một *từ máy* 8 bit là $2^7 - 1 = 127$, trong *từ máy* 16 bit là $2^{15} - 1 = 32767$, trong một *từ máy* 32 bit là $2^{31} - 1 = 2.147.483.647$.

b) Dữ liệu kiểu thực - Real

Một số thực bất kỳ có thể biểu diễn bằng tổng các số với mũ của 2:

$$X = \sum_{i=-m}^{i=n} a_i \times 2^i$$

trong đó $a_i = 0$ hoặc 1.

Có hai cách lưu số thực: dấu chấm tĩnh (fixed point) và dấu chấm động (floating point). Ví dụ 110.101 (6.625) hay 0.110101×2^3.

Số thực dấu chấm động trong ô nhớ được biểu diễn thành 2 thành phần: một phần của *từ máy* được dùng để lưu trữ một số cố định các bit của phần định trị, phần khác lưu trữ phần mũ.

<Phần định trị>.<phần mũ>

c) Dữ liệu kiểu ký tự - char

Máy lưu trữ *dữ liệu ký tự* dựa trên cơ sở gán mã số cho mỗi ký tự. Các ký tự được biểu diễn bằng mã nhị phân, mỗi *từ máy* 16 bit chia thành hai byte, mỗi byte lưu trữ một ký tự dưới dạng nhị phân. Ký tự được mã hoá theo ASCII (American Standard Code for information Interchange - Bộ mã chuẩn của Mỹ dùng để trao đổi thông tin) hay EBCDIC (Extended Binary Code Decimal Interchange Code - Mã BCD mở rộng). Phép toán phổ biến cho các ký tự là so lựa tuần tự (collating sequence).

Ví dụ: *ký tự* A trong bảng mã ASCII có mã là 65, *ký tự* B trong bảng mã ASCII có mã là 66.

Khi đó phép so sánh A>B sẽ cho kết quả sai.

d) Dữ liệu kiểu logic - Boolean

Mỗi giá trị logic được biểu diễn bởi một chữ số ở hệ đếm nhị phân là 0 hoặc 1. Trong các ngôn ngữ thông dụng có 4 phép toán logic là AND, OR, NOT, XOR.

Trên đây nhắc lại một số kiểu dữ liệu cơ bản, có trong mọi hệ thống máy tính, trong mọi loại ngôn ngữ lập trình.

Việc xác định kiểu dữ liệu là rất quan trọng, nó giải nghĩa cho một xâu bit dữ liệu và xác định các phép toán được sử dụng.

Tương ứng với mỗi kiểu dữ liệu chỉ có một số phép toán được áp dụng, ví dụ trong ngôn ngữ Pascal:

Với kiểu Integer chỉ dùng các phép: +, -, *, /, ^, mod, div

Với kiểu Real dùng các phép: +, -, *, /, ^ ;

Với kiểu Boolean dùng các phép: and, or, not, xor

Với kiểu Char dùng các phép: +, concat.

1.1.3. Cấu trúc dữ liệu - Data Structure

Định nghĩa: Cấu trúc dữ liệu là cách thức tổ chức dữ liệu trong bộ nhớ máy tính.

Trong các loại cấu trúc dữ liệu, người ta phân chia ra cấu trúc đơn giản, cấu trúc phức tạp, cấu trúc tĩnh, cấu trúc động.

- Cấu trúc dữ liệu đơn giản: là các kiểu dữ liệu nguyên thuỷ được định nghĩa trong các ngôn ngữ lập trình, ví dụ như Integer, Real, Boolean, Char trong ngôn ngữ Pascal; Int, Float, Char trong ngôn ngữ C.

Trong thực tế, có những bài toán mà cách tổ chức dữ liệu đơn giản không đáp ứng được, đòi hỏi phải tổ chức phức tạp hơn, ví dụ như một dãy số, một danh sách các dữ liệu, v.v...

- Cấu trúc dữ liệu tĩnh: là cách tổ chức mà kích thước của các phần tử dữ liệu là cố định, cấu trúc cũng cố định trong khi chương trình dịch làm việc. Trong ngôn ngữ Pascal, các kiểu array, kiểu dữ liệu liệt kê, hay các kiểu nguyên thuỷ như Integer, Real, ...

- Cấu trúc dữ liệu động: là cách tổ chức mà có thể thay đổi cả kích thước và cấu trúc của các dữ liệu. Các dữ liệu động này được sinh ra trong quá trình chương trình thực hiện. Chúng không được thể hiện khi dịch chương trình, và thường được sử dụng con trỏ để tạo ra dữ liệu động.

Ngoài các khái niệm trên, để phục vụ cho việc học tập, nghiên cứu và triển khai các ứng dụng, người ta còn chia ra: Cấu trúc dữ liệu tuyến tính, cấu trúc dữ liệu phi tuyến. Những cấu trúc mà khi xử lý được tiến hành ở dạng một dãy liên tục các dữ liệu thì được nhóm gộp vào loại tuyến tính, ví dụ như array, stack, queue, list đơn giản. Những cấu trúc còn lại được gọi chung là phi tuyến.

1.2. CÁC MÔ HÌNH DỮ LIỆU

Để đảm bảo hiệu quả các quá trình xử lý thông tin cần phải tổ chức dữ liệu cho phù hợp. Mô hình hoá dữ liệu phải phản ánh được thế giới hiện thực một cách tốt nhất và phải cài đặt được trong máy tính.

Từ Data (dữ liệu) có xuất xứ từ thuật ngữ "datum" tiếng Hy Lạp có nghĩa là "sự kiện". Tuy nhiên không phải bao giờ dữ liệu cũng tương ứng với một sự kiện cụ thể nào đó, hiện diện trong thế giới thực. Chúng ta gọi dữ liệu là mô tả của một hiện tượng bất kỳ (hay là một ý tưởng) mà đáng giá để biểu diễn nó và xác định nó chính xác. Từ xa xưa con người đã sử dụng nhiều loại phương tiện khác nhau để ghi nhận các sự kiện, ý tưởng: đá, giấy,... thông thường dữ liệu (các sự kiện - Data) và diễn giải nó (ngữ nghĩa - semantic) được ghi cùng nhau vì ngôn ngữ tự nhiên khá tinh tế, phong phú để biểu diễn hiện tượng, sự kiện. Khi ta nói " dung lượng đĩa cứng 40MB" thì ở đây 40 là dữ liệu, còn ngữ nghĩa "dung lượng MB". Trong một số trường hợp thì dữ liệu và ngữ nghĩa (semantic) bị tách rời (ví dụ: bảng giờ tàu - diễn giải ở bên trên, còn phía dưới là giờ chạy của các tàu ở các tuyến đường).

Trong tin học thì dữ liệu và ngữ nghĩa càng bị phân tách. Trong bộ nhớ của máy tính điện tử, người ta chỉ làm việc với dữ liệu, không để ý đến ngữ nghĩa, phần ngữ nghĩa của dữ liệu bản thân người lập trình phải ngầm hiểu, lưu giữ ở bộ nhớ riêng. Thường trong chương trình phải ghi chú về ý nghĩa của các loại dữ liệu, thiếu ghi chú này thì dữ liệu chỉ là các dãy bit đơn thuần.

Sự linh hoạt của biểu diễn dữ liệu đạt được nhờ hai phương pháp:

- Có nhiều cách nhìn khác nhau đối với cùng một đối tượng dữ liệu;
- Biểu diễn đồng nhất hoá các dữ liệu khác nhau.

Ví dụ: với đối tượng là *Con người*, ta có hai cách tiếp cận:

Theo phương pháp thứ nhất: cùng một đối tượng con người, nhưng trong danh sách lớp học đó là họ tên sinh viên, trong bài toán xét lên lương đó là họ tên cán bộ công nhân viên, trong quản lý bệnh viện đó là họ tên bệnh nhân...

Theo phương pháp thứ hai, ta có thể coi mọi người từ Giám đốc, trưởng phó phòng, đến nhân viên... đều là cán bộ công nhân viên.

Đó chính là những lý do dẫn đến cần phải trừu tượng hoá dữ liệu. Phương tiện để biểu diễn dữ liệu gọi là Các mô hình dữ liệu (Data model). Mô hình dữ liệu là phương tiện để trừu tượng hoá dữ liệu, cho khả năng nhìn thấy "rừng" (nội dung thông tin của dữ liệu) chứ không chỉ "từng cây riêng rẽ" (các giá trị riêng lẻ của dữ liệu). Mô hình dữ liệu cho khả năng biểu diễn một phần ngữ nghĩa (sematic) của dữ liệu. Mô hình dữ liệu xác định các quy tắc mà theo đó sẽ cấu trúc hoá dữ liệu.

Tập hợp các dữ liệu mà có cấu trúc tương ứng với một sơ đồ dữ liệu nhất định thì gọi là một cơ sở dữ liệu.

Những mô hình dữ liệu cơ bản đã được nghiên cứu nhiều là:

- Mô hình phân cấp (thứ bậc - Hierarchical Model)
- Mô hình mạng lưới (Network Model)
- Mô hình quan hệ (Relational Model)
- Mô hình đối tượng/quan hệ (Object/Relational Model)
- Mô hình hướng đối tượng (Object - Oriented Model
- Mô hình nửa cấu trúc (Semistructured Model)
- Mô hình hiệp hội (Associative Model)
- Mô hình thực thể - thuộc tính - giá trị (Entity - Attibute - Value (EAV) Model)
- Mô hình ngữ cảnh (Context Model).

Dưới đây ta sẽ xem xét một số mô hình, thứ tự được trình bày theo mức độ phổ dụng.

1.2.1. Mô hình quan hệ - Relational Model

Mô hình dữ liệu quan hệ do Edgar F. Codd đề xướng những năm 1960.

Edgar F. "Ted" Codd sinh ngày 23 tháng 8 năm 1923 ở Portland, Dorset, nước Anh. Ông tốt nghiệp Khoa Toán và Hoá học ở Đại học Exeter, Oxford. Năm 1948 chuyển đến New York làm việc cho Công ty máy tính IBM với tư cách là lập trình viên phần thuật toán. Năm 1952 Ông chuyển đến Ottawa - năm 1962 lại quay về Mỹ và làm bằng Tiến sỹ Khoa học Máy tính ở Trường Đại học Michigan ở Ann Arbor. Năm 1964, Ông chuyển đến làm việc ở Trung tâm nghiên cứu Almaden của Hãng IBM ở San Jose California. Những năm 1960, 1970 Ông nghiên cứu lý thuyết về cấu trúc dữ liệu. Năm 1970, Ông cho

đăng bài báo nổi tiếng "Mô hình dữ liệu quan hệ cho các Ngân hàng dữ liệu cỡ lớn - A Relational Model of Data for Large Shared Data Banks". E.F.Codd có rất nhiều đóng góp cho ngành Công nghệ thông tin, nhiều hệ quản trị dữ liệu nổi tiếng hiện nay như Oracle, FoxPro,... đều xuất phát từ lý thuyết của Ông. Người ta đã lấy tên Ông đặt cho một dạng chuẩn dữ liệu: Dạng chuẩn Boyce-Codd. Ông được nhận giải thưởng Turing năm 1981. Edgar F. Codd mất ngày 18 tháng 4 năm 2003 tại Williams Island, Bang Florida, Mỹ, thọ 79 tuổi).

Edgar F. Codd

Phương tiện duy nhất để cấu trúc hoá dữ liệu trong mô hình quan hệ là quan hệ (Relation). Các quan hệ được hiểu theo ý nghĩa của toán tập hợp và được thể hiện dưới dạng các bảng. Mô hình dữ liệu dựa trên các quan hệ và được biểu diễn bằng các bảng lần đầu tiên được E.F. Codd đề xướng trong tài liệu "A relational model of Data for large shared Data banks". Commun. ACM, 13, p.377-387.

Một quan hệ được định nghĩa như sau:

Cho các tập hợp D_1, D_2, ..., D_n (không nhất thiết phải khác nhau), khi đó R là một quan hệ, được cho trên các tập hợp này, nếu R- là một tập hợp các corteges n-địa phương hay đơn giản là tập hợp các corteges mà trong mỗi cortege đó phần tử thứ nhất thuộc D_1, phần tử thứ hai thuộc D_2,... Tập hợp D_1 gọi là các Domen của R. Số n được gọi là bậc của R, số lượng corteges là lực lượng của R.

Mô hình dữ liệu quan hệ là dạng mô hình bảng - mỗi bảng là một quan hệ. Mở rộng cơ sở dữ liệu quan hệ là tập hợp các bảng. Các cột của bảng gọi là các thuộc tính, mỗi hàng của bảng tương ứng với một cortege của quan hệ. Để quản lý một cơ sở dữ liệu cần xây dựng một hệ quản trị. Trong các hệ quản trị cơ sở dữ liệu quan hệ, các thuộc tính còn được gọi là các trường - field; các hàng là các bản ghi - record.

Ví dụ: Có 5 bảng - hồ sơ thí sinh, báo danh-phách, phach1-diem1, phach2-diem2, phach3-diem3.

Hồ sơ thí sinh

Số BD	Họ tên	Ngày sinh	Đối tượng

Báo danh phách

Số BD	Phach1	Phach2	Phach3

Phach1-diem1

Phach1	Điểm 1

Phach2-diem2

Phach2	Điểm 2

Phach3-diem3

Phach3	Điểm 3

Mỗi thí sinh có 1 số báo danh duy nhất.

Mỗi số báo danh chỉ có duy nhất một phách môn1, một phách môn 2, một phách môn 3. Cả 3 số phách của 1 số báo danh không được trùng nhau. Mỗi phách 1 có 1 điểm môn 1, mỗi phách 2 có 1 điểm môn 2, mỗi phách 3 có 1 điểm môn 3.

Từ các bảng này ta sẽ kết nối và truy xuất ra được kết quả thi tuyển của từng thí sinh, đảm bảo không nhầm lẫn.

Các thao tác cơ bản đối với mô hình dữ liệu quan hệ là:

- Trích dữ liệu từ quan hệ để tạo một quan hệ mới theo điều kiện
- Cập nhật thông tin
- Chèn, xóa các trường
- Chèn, xóa các bản ghi
- Sắp xếp các trường theo một trật tự nào đó
- Tìm kiếm thông tin theo các điều kiện

Hiện nay, các hệ cơ sở dữ liệu được phổ cập phần lớn được thiết kế theo mô hình quan hệ, nổi bật là FOXPRO, ACCESS, ... Các hệ quản trị đi kèm theo là Visual Foxpro, Visual Basic, các ngôn SQL...

1.2.2. Mô hình mạng lưới

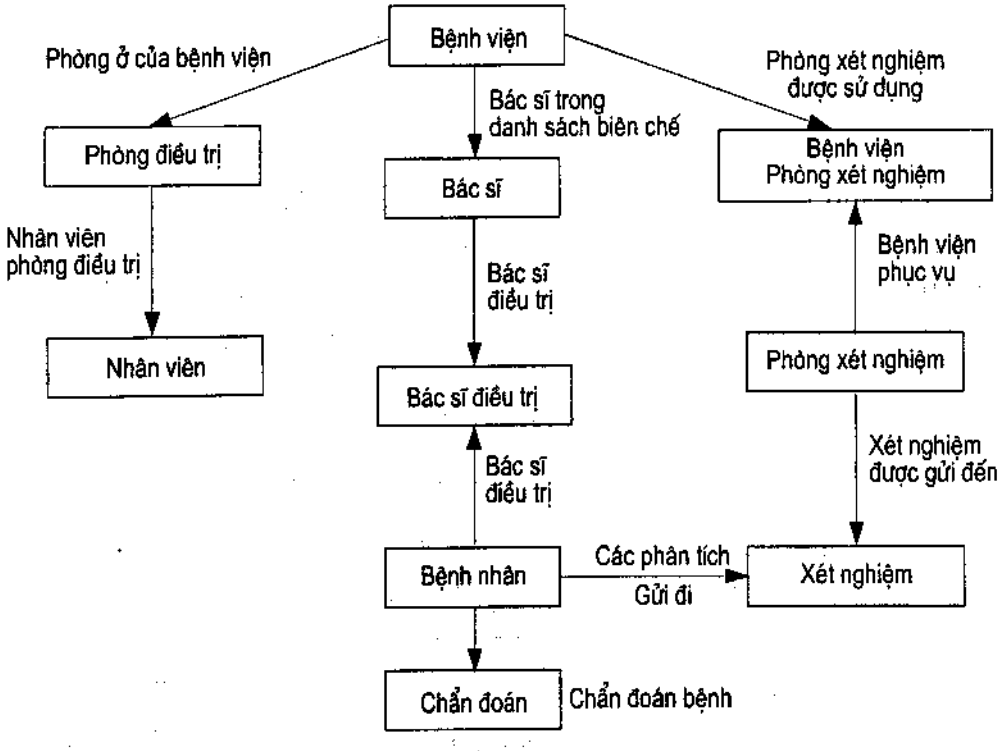
Trong thực tế, có nhiều bài toán có thể mô phỏng dưới dạng một mô hình có nhiều mối quan hệ ngang dọc kiểu mạng lưới. Năm 1971, Hội nghị về các ngôn ngữ Hệ thống Dữ liệu (The Conference on Data Systems Languages - CONDASYL) đã định nghĩa mô hình mạng lưới. Cơ sở để mô hình hóa mạng lưới là cấu trúc tập hợp. Một tập hợp bao gồm 1 kiểu bản ghi chủ, 1 tên tập hợp, và 1 kiểu bản ghi thành viên. Kiểu bản ghi thành viên (mức con) có thể tham gia trong nhiều tập hợp, nghĩa là cho phép có nhiều bản ghi ở mức trên (mức cha). Đến lượt mình, bản ghi chủ cũng có thể là bản ghi thành viên hay bản ghi con trong các tập hợp khác. Mô hình dữ liệu kiểu mạng CONDASYL dựa trên lý thuyết tập hợp của Toán học.

Hai khái niệm quan trọng ở đây là bản ghi và mối liên hệ.

Các kiểu bản ghi dùng để biểu diễn bảng các kiểu đối tượng.

Ví dụ: Để xây dựng phần mềm quản lý hệ thống các bệnh viện, ta dùng mô hình mạng lưới để mô tả.

Một sơ đồ dữ liệu cấu trúc mạng như sau:



Mỗi bản ghi có các trường dữ liệu riêng:

- Bệnh viện (*mã bệnh viện, chức năng điều trị, địa chỉ, điện thoại, số lượng giường*)

- Phòng điều trị (*mã phòng, chức năng điều trị, số lượng giường*)

- Người làm việc (*số hiệu, họ tên, chức vụ, ca trực, mức lương*)

- Bác sĩ (*số hiệu, số đăng ký*)

- Bệnh nhân (*số đăng ký, số giường, họ tên, địa chỉ, ngày sinh, nam/nữ, tên bệnh điều trị*)

- Chẩn đoán (*mã chẩn đoán, kiểu chẩn đoán, mức trầm trọng, và phòng ngừa*)

- Bệnh viện - phòng xét nghiệm (*mã bệnh viện, số liệu phòng xét nghiệm*)

- Phòng xét nghiệm (*số hiệu phòng xét nghiệm, tên gọi, địa chỉ và điện thoại*)

- Xét nghiệm (*mã xét nghiệm, kiểu, ngày ấn định, giờ ấn định, số phương án, tình trạng*)

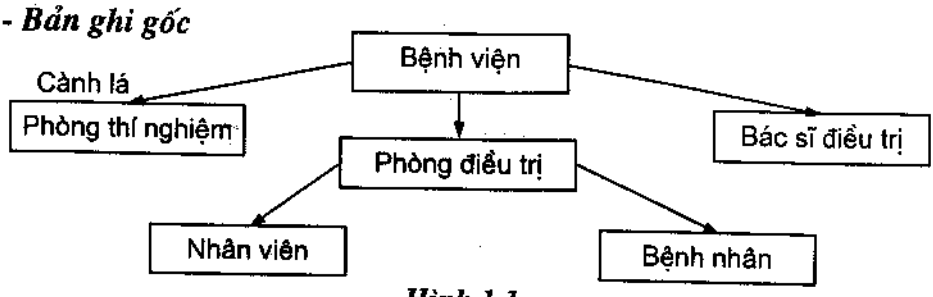
Trong sơ đồ dữ liệu kiểu mạng, có kiểu bản ghi chủ và bản ghi phụ thuộc (thành viên) ví dụ: *Bệnh viện* là bản ghi chủ và *Phòng điều trị* - phụ thuộc.

Hệ quản trị dữ liệu dạng mạng được cài đặt nổi tiếng là Condasyl Data Base Task Group (DBTG) (1971).

1.2.3. Mô hình phân cấp - thứ bậc (Hierarchical model)

Trong mô hình phân cấp dữ liệu được tổ chức dạng cây, cũng là một dạng của mô hình có kiểu đồ thị với các đỉnh là các bảng. Hệ quản trị dữ liệu dạng phân cấp nổi tiếng nhất là họ IMS (Information Management System/Virtual Storage) khi xây dựng dự án đổ bộ lên mặt trăng (Apollo) của Mỹ trong những năm 1960 - 1970. Trong sơ đồ phân cấp, toán đồ cấu trúc là một cây được sắp trật tự (hình 1.1).

- Bản ghi gốc



Hình 1.1

- Bệnh viện (*mã bệnh viện, tên, địa chỉ, điện thoại, số giường*)

- Phòng xét nghiệm (*số hiệu phòng xét nghiệm, tên gọi, địa chỉ, điện thoại*)

- Phòng điều trị (*mã phòng, tên gọi, số giường*)

- Nhân viên (*số hiệu nhân viên, họ tên, chức vụ*)

- Bệnh nhân (*số đăng ký, số giường, họ tên, địa chỉ, ngày sinh, nam/nữ, bệnh án*)

- Bác sỹ (*số hiệu bác sỹ, họ tên, chuyên môn*).

Trong sơ đồ này có bản ghi cha, bản ghi con. Dữ liệu là 1 dãy các bản ghi, mỗi bản ghi chứa các trường giá trị. Mô hình dữ liệu sẽ tập hợp tất cả các thành phần dữ liệu thành các bản ghi. Các kiểu bản ghi này tương đương các bảng trong mô hình dữ liệu quan hệ, mỗi bản ghi riêng rẽ tương đương 1 dòng trong bảng. Để tạo các quan hệ giữa các kiểu bản ghi, mô hình thứ bậc dùng quan hệ - Cha - Con.

1.2.4. Mô hình Đối tượng/Quan hệ

Đây là một sự mở rộng của các Hệ thống quản trị dữ liệu dạng đối tượng - quan hệ (ORDBMSs), chúng cho phép lưu trữ thêm đối tượng mới vào hệ thống quan hệ trong trung tâm của các hệ thống thông tin hiện đại. Những tiện ích mới này đã cho phép tích hợp việc quản lý các cơ sở dữ liệu truyền thống, các đối tượng phức tạp như các dãy thời gian, các dữ liệu về vũ trụ, các dữ liệu đa phương tiện như âm thanh, hình ảnh, phim, các applets (là những chương trình viết bằng ngôn ngữ Java mà có thể nhúng vào trang HML). Bằng các phương pháp tổ hợp, đóng gói với các cấu trúc dữ liệu, một máy chủ ORDBMS có thể thực hiện các thao tác xử lý dữ liệu phức tạp đối với các dữ liệu đa phương tiện hay các đối tượng phức tạp khác.

Mô hình đối tượng/quan hệ đã tích hợp các ưu việt của mô hình dữ liệu dạng quan hệ và tính mềm dẻo của mô hình định hướng đối tượng. Kỹ sư thiết kế cơ sở dữ liệu có thể làm việc với cấu trúc dạng bảng dữ liệu quen thuộc và với ngôn ngữ định nghĩa dữ liệu DDL (Data Definition Language) khi xử lý các khả năng quản lý đối tượng mới.

Những ngôn ngữ xử lý ORDBMS là các ngôn ngữ quen thuộc như SQL, các ODBC (Open Data Base Connectivity), JDBC (Java Data Base Connectivity).

1.2.5. Mô hình định hướng đối tượng

Hệ Cơ sở dữ liệu định hướng đối tượng OODB (Object-Oriented Data Base) là tổ hợp của hệ thống ngôn ngữ lập trình hướng đối tượng và hệ thống lưu trữ dữ liệu. Sức mạnh của OODB có được nhờ việc ít phải xử lý dữ liệu đang lưu trữ cũng như dữ liệu được truyền đến hay dữ liệu đang xử lý trong các chương trình.

Trong Hệ quản trị Cơ sở dữ liệu quan hệ, các cấu trúc dữ liệu phức tạp được trải ra thành các bảng dữ liệu hoặc kết hợp các bảng lại theo cấu trúc bộ nhớ, ngược lại, trong hệ quản trị dữ liệu đối tượng không thực hiện việc lưu trữ hay khôi phục các đối tượng liên kết trong của trang web hay trong mô hình thứ bậc. Với sự sắp xếp 1-1 giữa các đối tượng của ngôn ngữ lập trình hướng đối tượng với các đối tượng của cơ sở dữ liệu sẽ có 2 lợi ích hơn so với các cách lưu trữ khác: nó cho phép thực hiện việc quản lý các đối tượng hoàn hảo hơn, đồng thời nó cho phép quản lý tốt hơn các quan hệ phức tạp giữa các đối tượng. Những tính ưu việt này của hệ quản trị cơ sở dữ liệu hướng đối tượng đã làm cho nó hỗ trợ tích cực hơn cho các phần mềm ứng dụng như phần mềm phân tích rủi ro tài chính của công ty, phần mềm dịch vụ viễn thông, cấu trúc tài liệu WEB, hệ thống tự động hoá thiết kế và sản xuất, v.v...

1.2.6. Mô hình nửa cấu trúc

Trong mô hình dữ liệu nửa cấu trúc, thông tin thường được liên kết với một sơ đồ mà nó chỉ tường minh thông qua bản thân dữ liệu, đôi khi còn gọi là "tự mô tả". Trong cơ sở dữ liệu loại này, không có sự phân biệt rõ ràng giữa dữ liệu và sơ đồ, mức độ cấu trúc của dữ liệu phụ thuộc vào phần mềm ứng dụng.

1.2.7. Mô hình dữ liệu liên kết

Mô hình dữ liệu liên kết chia các vật thể của thế giới thực mà dữ liệu của chúng được ghi dưới 2 dạng: các thực thể - đó là các đối tượng tồn tại một cách rời rạc, độc lập; các mối liên kết - là những thứ tồn tại phụ thuộc vào một hay nhiều vật thể khác. Một cơ sở dữ liệu liên kết bao gồm 2 cấu trúc dữ liệu:

- Một tập các phần tử, mỗi phần tử có một tên ký hiệu riêng - ID, một tên và một kiểu.

- Một tập các liên kết, mỗi liên kết có một ký hiệu riêng - ID, kèm theo 3 ID thể hiện 3 đối tượng khác là *đối tượng nguồn, động từ, đối tượng đích* (kết quả). Một trong 3 ID này cũng có thể là liên kết hay là phần tử.

Ngoài các mô hình dữ liệu trên, hiện có thêm 2 mô hình mới là *Mô hình Thực thể - Thuộc tính - Giá trị (EAV) và mô hình dữ liệu ngữ cảnh - Context Model).*

Chương 2
CẤU TRÚC DỮ LIỆU TUYẾN TÍNH

2.1. MẢNG - ARRAYS

2.1.1. Khái niệm mảng

Mảng là một dãy các phần tử dữ liệu cùng kiểu, được đặt chung 1 tên mảng và các phần tử được phân biệt bởi các chỉ số.

Phép toán cơ bản đối với mảng là truy nhập trực tiếp đến các phần tử của mảng để xử lý, lưu trữ hay đọc ra. Vì vậy mảng phải có một số cố định các phần tử, phải được sắp thứ tự. Truy nhập trực tiếp là đến thẳng phần tử dữ liệu, không theo một tuần tự nào, vì vậy thời gian truy nhập trực tiếp đến một phần tử bất kỳ là như nhau.

Mảng có tên mảng và danh sách chỉ số. Mảng phải được khai báo ở phần khai báo đầu chương trình. Trong Pascal khai báo mảng có dạng tổng quát như sau:

Type <tên kiểu mảng> = Array[<danh sách chỉ số>] of <kiểu phần tử>;

Ví dụ:
```
Const n = 50;
Type mang = array [1..n] of integer;
Var A, B: mang;
```

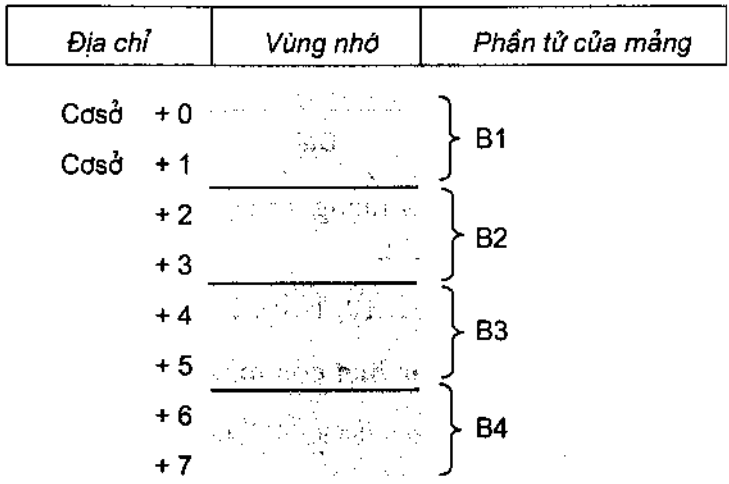
Từ khai báo mảng, bộ dịch sẽ dành vùng bộ nhớ cho mảng kể từ một địa chỉ cơ sở (base address). Gọi cơ sở (A) là địa chỉ cơ sở cho mảng A, khi đó địa chỉ của phần tử thứ a_i sẽ là:

Cơ sở (A) + (i-1).-

Nếu 1 phần tử mảng là 1 số nguyên, lưu trong bộ nhớ bằng một *từ máy* - word, thì địa chỉ của phần tử mảng chính là địa chỉ của từ máy đó.

Nếu một mảng số thực, mỗi phần tử được lưu trong hai *từ máy* thì việc tính địa chỉ để truy nhập phần tử mảng phải theo công thức:

Cơsở (B) + (i-1).2



Tổng quát: nếu mỗi phần từ mảng cần k từ máy thì phần tử thứ i của mảng được bắt đầu ở địa chỉ Cơ sở (B) + (i-1)k.

Trên đây là trường hợp chỉ số là đoạn con các số nguyên.

- Khi chỉ số là kiểu thứ tự chữ cái:

Ví dụ:
```
Type  Mang = array['A'..'Z'] of integer;
Var   M: mang;
      Chu: char;
```

Mỗi phần tử mảng M được chứa trong một từ máy. Khi đó, xác định phần tử mảng M[chu] như sau:

I = ord (chu) – ord('A')+1

Và địa chỉ của M[chu] là:

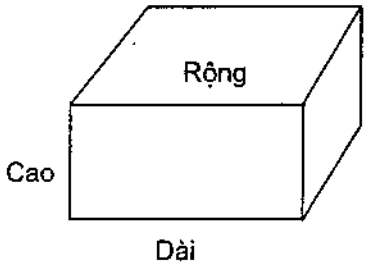
Cơsở (M) + (i-1) = Cơ sở (M) + ord(chu) – ord ('A')

- Mảng nhiều chiều:

Các ngôn ngữ bậc cao cho phép sử dụng mảng nhiều chiều. Trong Pascal không giới hạn số chiều của mảng:

Array [<chỉ số $_1$>, <chỉ số $_2$>, ...<chỉ số$_n$>] of <kiểu phần tử>

Ví dụ:



Mô tả kích thước một đối tượng không gian 3 chiều (chẳng hạn các kích thước đồ gỗ: tủ, giường...):

KT = array[1..10, 1..10, 1..10] of real;

2.1.2. Các trường hợp đặc biệt của mảng

a) Mảng của mảng

Trong Pascal có thể sử dụng mỗi phần tử mảng lại là một mảng.

```
Ví dụ:  Const    Cao = 10;
                 Dai = 10;
                 Rong = 10;
        Type     mang_cao = array [1..cao] of real;
                 Mang_dai = array[1..dai] of mang_cao;
                 kichthuoc = array [1..rong] of mang_dai;
```

Trong các bài toán thực tế, hay dùng nhất là mảng hai chiều. Dữ liệu của các mảng được lưu trữ trong các *từ máy* (word). Trong bộ nhớ máy tính các *từ máy* được sắp xếp một cách trình tự, tuyến tính, một chiều. Vì vậy chúng ta cần phải biểu diễn được sự tương đương của mảng dữ liệu một chiều trong cấu trúc dữ liệu với mảng 2 chiều của chương trình.

Giả sử có mảng:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

Mảng này được lưu trữ trong bộ nhớ bắt đầu từ địa chỉ coso(A), gọi là *Địa chỉ cơ sở của mảng*, các phần tử được bố trí theo hàng: a_{11} a_{12} a_{13} a_{14} a_{21} a_{22}... a_{34} a_{41}... a_{44}. Khi đó, địa chỉ của phần tử đầu của hàng thứ i được xác định theo công thức:

$$\text{Địa chỉ của phần tử } a_{i1} = \text{coso(A)} + (i-1) \times 4$$

Và địa chỉ của phần tử a_{ij} trong hàng này sẽ là:

$$\text{Cosở (A)} + (i-1) \times 4 + (j-1)$$

Ưu khuyết điểm của mảng:

- Ưu điểm: cho phép tạo các danh sách và tìm kiếm dễ dàng: mảng được sắp thứ tự.

- Khuyết điểm: khi khai báo mảng, phải chỉ ra kích thước tối đa song khi sử dụng có thể dùng không hết, vì vậy sử dụng mảng thường lãng phí một số ô nhớ.

b) Mảng là ma trận thưa (Sparse matrices)

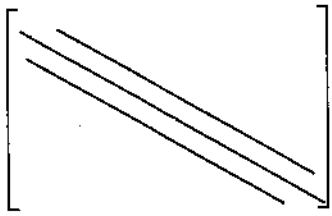
Trong các ngành kỹ thuật, ma trận thưa được sử dụng tương đối phổ biến, nó phù hợp để mô phỏng các bài toán kỹ thuật, đặc biệt trong ngành Xây dựng và Cơ khí. Có nhiều loại ma trận thưa, sau đây ta sẽ xét một số kiểu điển hình.

- *Kiểu 1*. Các phần tử bằng 0 nằm trên đường chéo chính - ta gọi là ma trận tam giác dưới, hoặc các phần tử bằng 0 nằm dưới đường chéo chính - ta gọi là ma trận tam giác trên.

$$\begin{vmatrix} a_{11} & a_{12} & \dots & \dots & a_{1n} \\ 0 & a_{22} & \dots & \dots & a_{2n} \\ 0 & 0 & a_{33} & \dots & a_{3n} \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & a_{nn} \end{vmatrix} \qquad \begin{vmatrix} a_{11} & 0 & \dots & \dots & 0 \\ a_{21} & a_{22} & 0 & \dots & 0 \\ a_{31} & a_{32} & a_{33} & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} \end{vmatrix}$$

- *Kiểu 2. Ma trận băng:*

Có các băng khác không: ma trận vuông trong đó các phần tử khác không nằm trên băng dựa theo đường chéo chính.



- *Kiểu 3: Ma trận thưa:* Thưa không theo quy luật nào

Để xử lý ma trận thưa thường dùng cách sau đây:

Nhập theo một bảng có 3 cột và (q+1) hàng, ở đây q là số phần tử khác không. Hàng thứ nhất là m, n, q (số hàng, số cột, số phần tử khác 0 của ma trận ban đầu). Những hàng tiếp theo là vị trí hàng cột của a[i, j] ≠ 0 và giá trị của phần tử a[i, j] (i, j, <giá trị>):

m	n	q
i	j	a[i.j] ≠ 0

Ví dụ: Cho ma trận thưa A có 7 hàng 6 cột, có 9 phần tử khác 0:

	1	*2*	*3*	*4*	*5*	*6*
1	23	0	0	0	46	33
2	0	14	8	0	0	0
3	0	0	0	9	0	0
4	0	0	0	0	0	0
5	85	0	0	0	0	0
6	0	0	–37	0	0	0
7	0	5	0	0	0	0

↓

Ta nhập theo mảng 10 hàng 3 cột:

A			
	[1	7	6	9
	[2	1	1	23
	[3	1	5	46
	[4	1	6	33
	[5	2	2	14
	[6	2	3	8
	[7	3	4	9
	[8	5	1	85
	[9	6	3	-37
	[10	7	2	5

c) Các phép toán với ma trận thưa

* *Hoán vị ma trận:* khi hoán vị, ta chuyển hàng thành cột và ngược lại.

b[ij] = a[ji] khi i ≠j ; b[ij] = a[ji] khi i = j

		1	*2*	*3*
	B[1	6	7	9
$B = A^T \Rightarrow$	B[2	1	1	23
	B[3	1	5	85
	B[4	2	2	14
	B[5	2	7	5
	B[6	3	2	8
	B[7	3	6	-37
	B[8	4	3	9
	B[9	5	1	46
	B[10	6	1	33

* *Thuật toán hoán vị ma trận thưa:*

Có thể hoán vị ma trận thưa theo một số cách.

- Thuật toán 1:

1. Tìm tất cả các phần tử ở cột 1 cho vào hàng 1.

2. Tìm tất cả các phần tử ở cột 2 cho vào hàng 2

.....

q. Tiếp tục cho đến hết

Như vậy, ta sẽ hoán vị giá trị ở cột 1 (giá trị I) và cột 2 (giá trị J), sau đó sắp xếp lại ma trận theo trật tự tăng dần cột 1 (sắp xếp theo cột).

- Thuật toán 2:

Ngay từ đầu ta đã xem xét và sắp xếp luôn trật tự các phần tử của mảng kết quả.

Chương trình sau đây thể hiện thuật toán dạng 1 ở trên.

```
program hvmt_thua;
 var   m,n,i,j,k,q:integer;
       aluu,b:array[1..50,1..3] of integer;
       tg:integer;
 begin
    writeln('Nhap ma tran A');
    write('So hang(m) :');readln(m);
    write('So cot (n) :');readln(n);
    write('So phan tu khac 0 (q) :');readln(q);
    aluu[1,1]:=m;aluu[1,2]:=n;aluu[1,3]:=q;
 for i:=2 to q+1 do
begin write('aluu[',i,'1]=');readln(aluu[i,1]) ;
     write('aluu[',i,'2]= ');readln(aluu[i,2]);
    write('aluu[',i,'3= ');readln(aluu[i,3]);
end;
Writeln(' hoan vi ma tran : ');
Writeln('Doi cot 1 <->2 : ');
   b[1,1]:=n;b[1,2]:=m;b[1,3]:=q;
   For i:=2 to q+1 do
            begin  b[i,1] := aluu[i,2];
                   b[i,2] := aluu[i,1];
```

```
                    b[i,3] := aluu[i,3];
                end;
        Writeln('Sap xep lai ma tran B theo cot 1: ');
        For i:=2 to q do
         For j:=i+1 to q+1 do
           If  b[i,1] > b[j,1] then
            For k:=1 to 3 do
                begin  tg:=b[i,k];
                    b[i,k]:=b[j,k];
                    b[j,k]:=tg
                end;
        Writeln('Hien ket qua hoan vi : ');
        for i:=1 to q+1 do
          begin
        for j:=1 to 3 do write('| ',b[i,j],' | ') ;
        writeln
         end;
        readln;
        End.
```

* *Nhân 2 ma trận thưa:*

Phép nhân ma trận có thể làm biến đổi tính chất "thưa" của ma trận kết quả.

- Tích hai ma trận thưa chưa chắc đã là ma trận thưa

Ví dụ:
$$\begin{bmatrix}1&0&0\\1&0&0\\1&0&0\end{bmatrix}\times\begin{bmatrix}1&1&1\\0&0&0\\1&0&0\end{bmatrix}=\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$$

* *Cộng hai ma trận thưa:* Phép cộng hai ma trận thưa cũng có thể làm cho ma trận kết quả trở thành ma trận không thưa, tuỳ theo dạng ma trận thưa mà ta xử lý:

Ví dụ: Đối với ma trận tam giác trên, khi đưa ra kết quả ta viết:

```
for i:=1 to n do
for j:=1 to n do
   if abs(i-j) >1 then C1[i,j] := ϕ
         else if abs (i-j) <=1 then begin
                                    C1[i,j]:=C[i,j-(i-2)];
                                    Write(C1[i,j]:8:2);
                                    end;
```

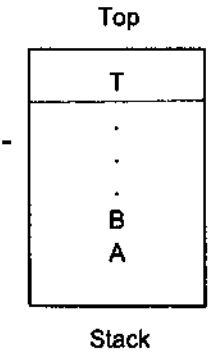
2.2. NGĂN XẾP - STACKS

2.2.1. Khái niệm

Ngăn xếp là một cấu trúc dữ liệu trừu tượng, xử lý theo kiểu vào sau ra trước (last in first out - LIFO) nó là một danh sách hay một dãy bản ghi trong đó mỗi phép toán thêm vào hay bớt đi đều thực hiện ở một đầu gọi là đỉnh ngăn xếp.

Ví dụ kiểu ngăn xếp:

1. Chuyển đổi cơ số
2. Xếp chồng đĩa và lấy ra dùng
3. Gọi và kết thúc các thủ tục



Các phép toán cơ bản liên quan đến ngăn xếp là:

1. Create(S) - Tạo S như là stack rỗng;
2. Add(i, S) - Thêm phần tử i vào S và cho stask mới;
3. Delete(S) - Bớt đi phần tử ở đỉnh ngăn xếp và tạo stask mới;
4. Top(S) - Tìm và lấy ra phần tử đỉnh ngăn xếp không rỗng
5. IsEmpty(s) - Cho giá trị true nếu S rỗng hay false nếu không rỗng.

Xét bài toán đổi một số ở hệ đếm cơ số 10 sang hệ 2 theo cách sử dụng ngăn xếp.

1. Nhập số cần dịch đổi, tạo một ngăn xếp rỗng.
2. Khi mà số <> 0 thì làm các việc:
   a) Tính Số dư: = Số mod 2;

b) Nhớ lấy số dư (gửi số dư vào đỉnh ngăn xếp);

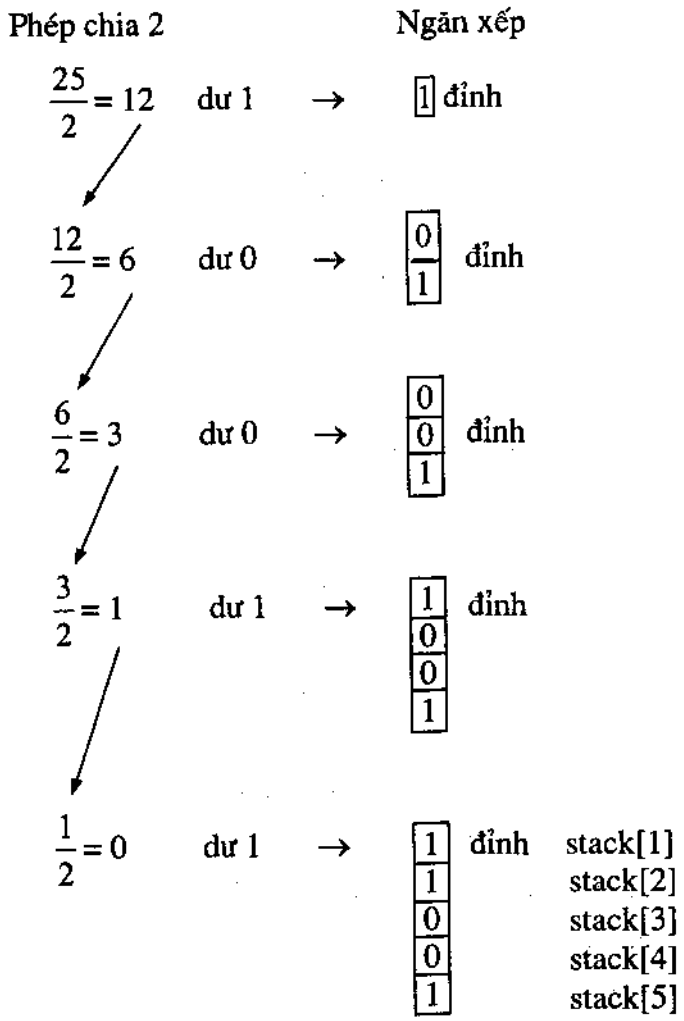
c) Thay số bằng giá trị phần nguyên của phép chia nguyên của số cho 2 (So: = so div 2).

3. Hiện kết quả: lần lượt hiện các ô nhớ chứa số dư, đi ngược từ cuối đến đầu, sẽ được biểu diễn hệ 2 của số thập phân ban đầu. Để làm việc này, trước hết kiểm tra, nếu ngăn xếp không rỗng thì:

a) Lấy số dư từ đỉnh ngăn xếp;

b) Hiện lên màn hình số dư vừa lấy.

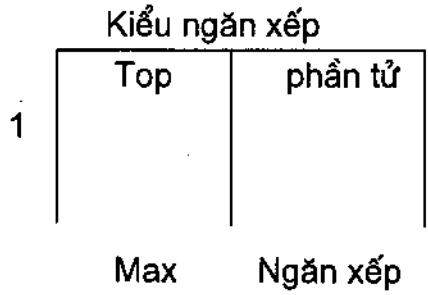
Ta có thể biểu diễn theo sơ đồ sau:



Cách đơn giản nhất để thể hiện stack là sử dụng mảng 1 chiều, chẳng hạn ký hiệu stack[1..n]. Phần tử đầu tiên (hay là đáy ngăn xếp) là stack[1], phần tử thứ 2 là stack[2], phần tử thứ i là stack[i]. Ta dùng 1 biến *top* để chỉ phần tử ở đỉnh ngăn xếp.

2.2.2. Sử dụng mảng và bản ghi để cài đặt ngăn xếp

Theo cách làm trên, khi thêm phần tử ở đỉnh ngăn xếp thì phải dịch chuyển các phần tử ở phía dưới đỉnh xuống 1 vị trí; khi lấy 1 phần tử thì phải dịch chuyển các phần tử lên 1 vị trí, điều này làm mất thời gian. Để khắc phục, ta sẽ lật ngược ngăn xếp, coi phần tử đầu tiên, stack[1] sẽ là đáy ngăn xếp, dùng thêm 1 biến phụ *top* để lưu phần tử ở đỉnh ngăn xếp. Ta sẽ dùng 1 mảng để lưu trữ các phần tử của ngăn xếp. Cách lưu này dẫn đến việc cần sử dụng bản ghi:



Khi đó ta có thể sử dụng ngăn xếp để viết các chương trình với một số khai báo như sau:

```
Const n = <1 số nguyên xác định>;
Type    Kieu_Ptu = <1 kiểu dữ liệu xác định>;
        Day_NganXep = array[1..n] of Kieu_Ptu;
        Kieu_NganXep = record
                           Top:0..n;
                           Phantu: Day_NganXep;
                end;
```

Ví dụ bài toán đổi cơ số 10 sang cơ số 2:

```
program doicoso;
 uses dos;
 const stacklimit=50;
```

```
type Kieu_Ptu=integer;
   Day_NganXep=array[1..stacklimit] of Kieu_Ptu;
   Kieu_NganXep=record
            top:0..stacklimit;
            phantu:Day_NganXep;
          end;
var so,sodu:integer;
  stack:Kieu_NganXep;
  traloi:char;
procedure thietlap(var stack :Kieu_NganXep);
     begin stack.top := 0 end;
function IsEmpty(stack :Kieu_NganXep):boolean;
  begin IsEmpty := (stack.top = 0) end ;
procedure Top(var stack :Kieu_NganXep; var Item: Kieu_Ptu);
 Begin if IsEmpty(stack) then halt
        else with stack do
          begin Item := Phantu[top];
              top := top -1
          end
 End;
 procedure Them(var stack :Kieu_NganXep;Item : Kieu_Ptu);
   begin if stack.top = stackLimit then halt
          else with stack do
            begin top := top + 1;
                phantu[top] := Item
            end
   end;
Begin (* chuong trinh chinh *)
 repeat
```

```
write('Dua vao so can doi :');readln(so);
thietlap(stack);
while so <> 0 do
  begin sodu := so mod 2;
      them(stack,sodu);
      so := so div 2
  end;
writeln('Bieu dien co so 2 :');
 while not IsEmpty(stack) do
   begin top(stack,sodu);
       write(sodu:1)
  end;
  readln;
  write('Co lam tiep khong ?(Y/N)');readln(traloi);
until not(upcase(traloi)='Y')
end.
```

2.2.3. Thư viện chứa các thao tác cơ bản trên ngăn xếp

Để sử dụng các thao tác cơ bản trên ngăn xếp như những hàm mẫu, ta tạo thư viện Stack.tpu như sau:

Gõ tệp ***Stack.pas*** này và lưu vào thư mục C:\tp\units:

```
UNIT STACK;
INTERFACE TYPE
StackElement=integer;(* có thể thay đổi*)
PointerType=^StackNode;
StackNode=record
                Du_Lieu:StackElement;
                Next:PointerType;
                end;
StackType= PointerType;
```

```
Procedure CreateS(Var Stack : StackType);
Function EmptyS(Stack : StackType):Boolean;
Procedure Pop(Var Stack:StackType;var Item:StackElement);
Procedure  Push(Var Stack:StackType; Item:StackElement);
IMPLEMENTATION
Procedure CreateS(Var Stack : StackType); (*Tạo ngăn xếp rỗng*)
 Begin  Stack := Nil end;
Function EmptyS(Stack : StackType):Boolean;
 Begin   EmptyS := (Stack = Nil) end ;
Procedure Push(Var Stack:StackType;Item:StackElement); (*Đẩy item
vào ngăn xếp*)
  Var TempPtr :PointerType;
 Begin
   New(TempPtr);
   TempPtr^.Du_Lieu := Item;
   TempPtr^.Next := Stack;
   Stack := TempPtr;
 End;
Procedure  Pop(Var Stack:StackType; Var  Item:StackElement);
(* Lấy item ra từ đỉnh ngăn xếp Stack*)
  Var   TempPtr :PointerType; (* TempPtr - con trỏ tạm thời chỉ đến
nút đỉnh Stack *)
 Begin
  IF   EmptyS(Stack) Then halt
         Else
                 Begin   Item := Stack^.Du_Lieu ;
                         TempPtr :=  Stack;
                  Stack := Stack^.Next;
                  Dispose(TempPtr);
                End ;
   End(*pop*);
END.
```

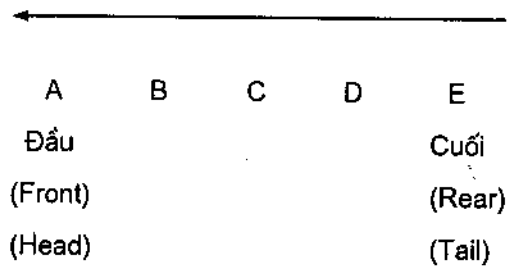
Sau đó khởi động Turbo Pascal, mở tệp Stack.pas, chọn menu dịch chương trình - Compile. Chú ý chọn Destination ⇒ Disk. Kết quả dịch sẽ cho tệp Stack.tpu.

2.3. HÀNG ĐỢI - QUEUE

2.3.1. Khái niệm

Trong thực tế ta gặp nhiều trường hợp phải xử lý thông tin theo kiểu phải xếp hàng: xếp hàng chờ giải quyết công văn giấy tờ, xếp hàng vào làm thủ tục ở Cảng hàng không, xếp hàng mua hàng hoá tại Cửa hàng... Ai đến trước được giải quyết, phục vụ trước, ai vào sau được phục vụ sau (First in - First out. FIFO), khác với Stack là vào sau - ra trước (LIFO).

Như vậy, Hàng đợi là một danh sách dữ liệu, trong đó các phép chèn, thêm phần tử được thực hiện ở một đầu danh sách (Front), còn các phép xoá được thực hiện ở đầu kia (điểm cuối - Rear).



Các nhiệm vụ tính toán, in ấn được xếp hàng chờ đợi xử lý do hệ điều hành quản lý là một ví dụ về hàng đợi được áp dụng trong tiến trình xử lý của máy tính.

Các phép toán cơ bản khi xử lý hàng đợi là:

CreateQ(Q): Khởi tạo hàng đợi;

AddQ(i, Q): Thêm phần tử i vào cuối hàng đợi;

DeleteQ(Q): Lấy ra khỏi đầu hàng đợi một phần tử;

IsEmptyQ(Q): xác định hàng đợi có rỗng không;

Front(Q): Cho phần tử đầu hàng đợi.

Hàng đợi khá giống ngăn xếp, chúng ta có thể dùng mảng để cài đặt hàng đợi. Ngoài một mảng d[1..n] cần dùng thêm 2 biến ***front*** (*đầu hàng*) và ***rear*** (*cuối hàng*).

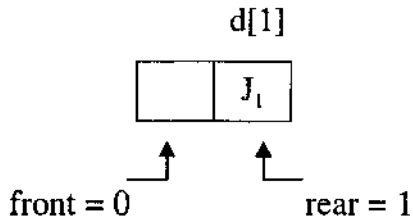
front luôn chỉ phần tử đầu tiên của hàng đợi mà có thể lấy ra được, còn *rear* luôn chỉ vị trí cuối cùng, nơi có thể thêm vào một phần tử.

1) front ↓ ↓ rear

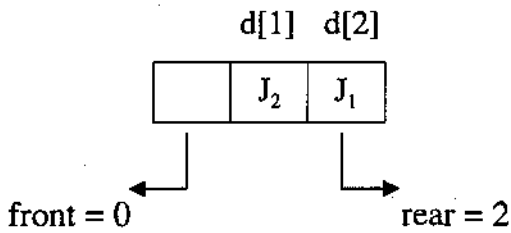
Hàng đợi rỗng, mới khởi tạo xong

front = rear = 0

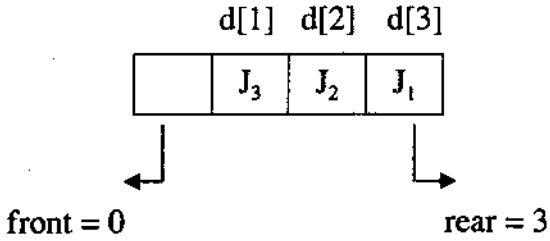
2) Đưa công việc J_1 vào hàng đợi:



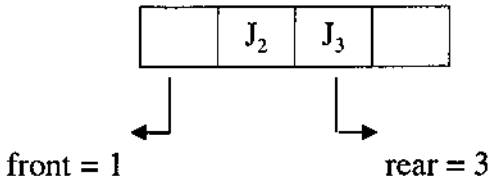
3) Công việc J_2 vào hàng đợi:



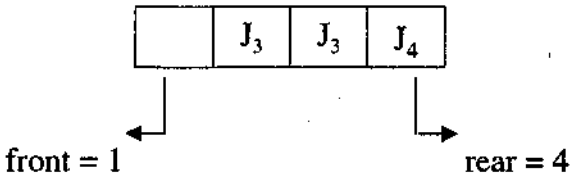
4) Công việc J_3 vào hàng đợi:



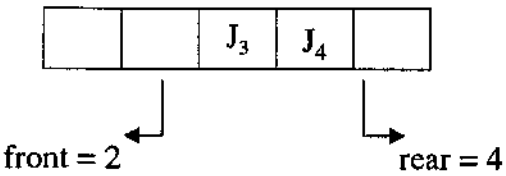
5) Công việc J_1 ra khỏi hàng đợi:



6) Công việc J_4 vào hàng đợi:



7) Công việc J_2 ra khỏi hàng đợi:



Trong cấu trúc này, mỗi lần lấy một công việc ra khỏi hàng đợi là phải dịch sang trái 1 vị trí. Ta có thể mô tả lại cách biểu diễn trên như sau:

• •		Q[1]	[2]	[3]	[4]	[5]	[6]	Giải thích
front	rear							
0	0		Hàng	đợi	rỗng			Khởi tạo Q
0	1	J_1						$J_1 \rightarrow Q$
0	2	J_1	J_2					$J_2 \rightarrow Q$
0	3	J_1	J_2	J_3				$J_3 \rightarrow Q$
1	3		J_2	J_3				J_1 ra khỏi Q
1	4		J_2	J_3	J_4			$J_4 \rightarrow Q$
2	4			J_3	J_4			J_2 ra khỏi Q

Với sơ đồ này ta có thể cài đặt mảng cho hàng đợi như sau:

- Thủ tục thiết lập hàng đợi.

```
Procedure CREATEQ(var Q: array[1..n] of item ; {n là const}
                  front, rear: 0..n;
        Begin front: = 0; rear: = 0 end;
```

- Kiểm tra hàng có rỗng không:

```
Function ISEMPTY(Q);
        Begin
                ISEMPTY: = ( front = rear);
        End;
```

- Cho phần tử ở đầu của hàng đợi:

```
Procedure FRONT(Q);
        Begin if ISEMPTY then Halt
                Else front: = front +1
        End;
```

- Thủ tục thêm phần tử vào hàng đợi:

```
Procedure ADDQ(item: Kieu_Ptu);
        Begin if rear = n then Writeln(' Hàng đầy ');
                rear: = rear +1;
                Q[rear]: = item;
        End;
```

- Lấy phần tử ra khỏi hàng đợi:

```
Procedure DELETEQ(var item: Kieu_Ptu);
        Begin if front = rear then (* Hàng rỗng*) Halt;
                front: = front +1;
                Item: = Q[rear];
        End;
```

2.3.2. Thư viện hàng đợi

Để tạo thư viện Queue.tpu ta cũng làm như đối với Stack. Gõ tệp Queue.pas sau đó lưu vào thư mục C:\tp\units.

```
UNIT QUEUE;
NTERFACE TYPE
QueueElement=integer;(* có thể thay đổi*)
QueuePtr=^QueueNode;
QueueNode=record
                    Du_Lieu:QueueElement;
                    Next:QueuePtr;
                    end;
QueueType= record
                    front,Rear :QueuePtr;
                    end;
Procedure CreateQ(Var Queue : QueueType);
Function EmptyQ(Queue : QueueType):Boolean;
Procedure AddQ(Var Queue:QueueType;Item:QueueElement);
Procedure  RemoveQ(Var Queue:QueueType; Var
Item:QueueElement);

IMPLEMENTATION

Procedure CreateQ(Var Queue : QueueType);
 Begin  Queue.Front := Nil; Queue.Rear := Nil;  end;
 Function EmptyQ(Queue : QueueType):Boolean;
 Begin   EmptyQ := (Queue.Front = Nil) and (Queue.Rear = Nil) end ;

Procedure AddQ(Var Queue:QueueType;Item:QueueElement);
  Var   Pt :QueuePtr;
 Begin
   New(Pt);
   Pt^.Du_Lieu := Item;
   With  Queue  Do
       If  EmptyQ(Queue)  Then
```

```
            Begin  Front := Pt; Rear := Pt  End
                Else
            Begin   Rear^.Next := Pt;    Rear := Pt   end;
End;

Procedure  RemoveQ(Var Queue:QueueType; Var
Item:QueueElement);
  Var   Pt :QueuePtr;
 Begin
  IF   EmptyQ(Queue) Then Writeln(' Hang doi rong ')
        Else   WITH  Queue DO
                  Begin   Item := Front^.Du_Lieu  ;
                          Pt :=  Front ;
                       IF Front <> Rear   THEN  Front := Front^.Next
                          Else Begin Front := Nil; Rear := Nil  end;
                     Dispose(Pt);
                  End ;(* WITH *)
   End(*REMOVEQ*);
END.
```

Sau đó vào Turbo Pascal, chọn menu dịch - compile, cho tệp kết quả là Queue.tpu, chú ý để Destination là Disk.

Chú ý:

1. Trong các thư viện Stack.pas và Queue.Pas có sử dụng các biến con trỏ (P) và biến động của con trỏ (p^) sẽ xét ở phần sau.

2. Dùng mảng để lưu trữ hàng đợi cũng sẽ gặp trở ngại khi thêm phần tử mới, cả hàng phải dịch chuyển sang phía trái, sang phải, tính lại địa chỉ các phần tử.

Ví dụ áp dụng: Dùng các phép toán cơ bản trên hàng đợi và ngăn xếp để viết thủ tục đảo ngược các phần tử của hàng đợi.

Bài giải:

```
(*  Giả thiết đã tạo 2 thư viện Stack.tpu và Queue.tpu *)
Uses STACK,QUEUE;
```

```
Procedure TaoHangDoi(var Q:QueueType);
 var  i:integer;
 begin  CreateQ(Q);
    Writeln('Dua vao cac so nguyen :');
    While not eoln Do
       begin read(i);AddQ(Q,i) end; readln;
 end;
 Var Q:QueueType;
    S:StackType;
    x:Integer;
 Begin TaoHangDoi(Q);
     CreateS(S);
    While  not EmptyQ(Q)  Do
       begin RemoveQ(Q,x); Push(S,x) end;
    While not EmptyS(S) Do
      begin  Pop(S,x);AddQ(Q,x);write(x,' ') end;
    readln;
 end.
```

2.4. DANH SÁCH - LIST

2.4.1. Khái niệm

Ngăn xếp và hàng đợi là những trường hợp đặc biệt của danh sách.

Một danh sách là 1 dãy hữu hạn (có thể rỗng) các phần tử.

Những thao tác đối với danh sách bao gồm:

1. Tạo một danh sách rỗng
2. Xác định danh sách có rỗng không
3. Duyệt, xử lý các phần tử thuộc danh sách
4. Thêm phần tử mới vào danh sách
5. Xoá một phần tử từ danh sách

Có thể khai báo danh sách như sau:

```
Const
        Max  = <số phần tử lớn nhất của dãy >;
```

```
Type  Kieu_Ptu  = < kiểu dữ liệu>;
      DanhSach_Day = array[1..Max] of  Kieu_Ptu;
      Kieu_Dsach  =  Record
                        Size :0... Max;
                        Phan_tu : DanhSach_Day;
                  End;
  Var   List : Kieu_Dsach;
```

Các thao tác 1, 2, 3 đối với danh sách thì đơn giản.

Đối với các thao tác chèn xoá các phần tử thì cần thực hiện các thủ tục dịch chuyển các phần tử.

- Thủ tục chèn một phần tử (Item) vào sau 1 vị trí xác định (pos)

```
Procedure Insert(var List: Kieu_Dsach; Item: Kieu_Ptu; pos: integer);
 Var i: integer;
Begin
    If Listfull(List) then Halt
          Else
        With List Do
           Begin
           For i:= size to Pos+1 do Phan_tu [i+1] := Phan_tu[i];
                    Phan_tu[Pos+1] := Item; size := size +1;
       End;
    End;
```

- Thủ tục xóa phần tử

```
Procedure Delete(var List: Kieu_Dsach; Pos: integer);
(* Xoá Item ở vị trí Pos *)
var i:integer;
Begin
 If  ListEmpty(List) Then  halt
   Else  With List  do
```

```
        Begin   size :=size -1;
                For i:= Pos to Size do Phan_tu[i]:= Phan_tu[i+1]
        end;
End;
```

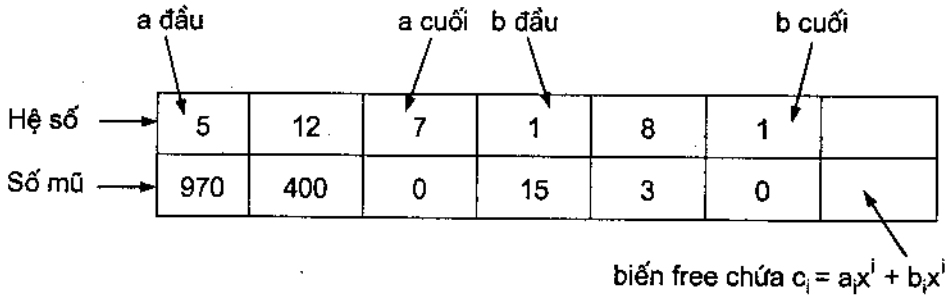
2.4.2. Ví dụ áp dụng danh sách

Ví dụ: Xét bài toán cộng 2 đa thức thưa:

$$A = \Sigma a_i x^i = 5x^{970} + 12x^{400} + 7$$

$$B = \Sigma b_j x^j = x^{15} + 8x^3 + 1$$

Ta tổ chức dữ liệu như sau:



Để viết chương trình, ta tổ chức đa thức là một mảng các nút, mỗi nút có 2 trường là *hệ số - coef* - và *số mũ - exp*. Dữ liệu được bố trí trong mảng lần lượt hết đa thức A rồi đến đa thức B. Thêm thông tin về địa chỉ chứa hệ số của số hạng đầu tiên của đa thức A, hệ số của số hạng cuối cùng của đa thức A, tương tự, hệ số đầu của B, cuối của B, đầu của C, và thêm 1 biến free lưu địa chỉ ô tự do. Chương trình như sau:

```
Const maxterm =100;
Type Node = record
              coef : integer;
              exp : 0..maxint;
            end;
var term : array[1..maxterm] of node;
i,j,n,free:integer;
aB,aE,bB,bE,cB,cE:integer;
  function compare(a,b :integer) : char;
     begin
```

```
        if a < b then compare := '<'
          else if a=b then compare := '='
                    else compare := '>' ;
    end;
 Procedure Newterm(c:integer;hs:integer);
   label 9999;
   begin
          if free > maxterm then
          begin write(' too many terms '); goto 9999 end;
       term[free].coef :=c;
       term[free].exp :=hs;
       free := free+1;
  9999: end;
procedure  add(aBegin,aEnd,bBegin,bEnd:integer;
           var cBegin,cEnd: integer);

  var i,j,free:integer; c:integer;
  Begin  i:=aBegin; j:=bBegin;cBegin:= free;
     While (i <= aEnd) and (j <= bEnd) do
      case compare(Term[i].exp,term[j].exp) of
        '=':begin
             c:=term[i].coef+term[j].coef;
             if c <> 0 then Newterm(c,term[i].exp);
             i:=i+1;j:=j+1
            end;
        '<': begin Newterm(term[j].coef,term[j].exp);
                j:=j+1
             end;
        '>' : begin Newterm(term[i].coef,term[i].exp);
                i:=i+1
             end;
      end;(* case & while *)
 { Nếu hết B chỉ còn A(x)}
       While i <= aEnd Do
     begin Newterm(term[i].coef,term[i].exp);
```

```
        i:=i+1
    end;
{Nếu hết A(x) chỉ còn B(x)}
  While j <= bEnd do
   begin Newterm(term[j].coef,term[j].exp);
        j:=j+1
   end;
  cEnd := free+1;
end;

(* Chuong trinh chinh *)
 Begin
  write('Cho so luong terms :' );readln(n);
  for i:=1 to n do
    begin write(' cho cac he so ');readln(term[i].coef) end;
  for i:=1 to n do
     begin  write(' cho so mu ',i,':');readln(term[i].exp)end;

  write('nhap aBegin,aEnd,bBegin,bEnd,cBegin,free :');
  readln(aB,aE,bB,bE,cB,free);
  add(aB,aE,bB,bE,cB,cE);
 writeln('hien ket qua :');
  for i:=7 to 11 do write(term[i].coef:4,'|');writeln;
  for i:=7 to 11 do write(term[i].exp:4,'|');writeln;
 readln;
 end.
```

2.5. BÀI TẬP

2.5.1. Bài tập về mảng

1. Ma trận tam giác dưới là một ma trận vuông, trong đó các phần tử khác ϕ chỉ có thể nằm dưới đường chéo chính và các phần tử của đường chéo chính (tức là A[i, j] = 0 nếu i<j).

Để tiết kiệm ô nhớ, ta chỉ lưu các phần tử ở trong tam giác dưới trong các từ máy (word) liên tiếp nhau. Hãy lập công thức tính địa chỉ các phần

tử a_{ij}, i ≥j, giả sử mỗi phần tử a_{ij} lưu trữ trong 1 từ, lưu theo hàng ma trận, bắt đầu từ địa chỉ CosởB.

2. Cho ma trận tam giác trên, các phần tử khác không nằm trên đường chéo chính và bản thân đường chéo chính. Nhiệm vụ làm như bài tập 1, lập công thức tính địa chỉ cho a_{ij}, i <j.

3. Ma trận 3 đường chéo: a[ij] = 0 nếu ABS(i-j) = 1. Lập công thức tính địa chỉ nếu tổ chức ma trận lưu trữ là dạng n hàng 3 cột:

Cột 1	Cột 2	Cột 3
0	a_{11}	a_{12}
a_{21}	a_{22}	a_{23}
a_{32}	a_{33}	a_{34}
$a_{n-1,n-2}$	$a_{n-1,n-1}$	$a_{n-1,n}$
$a_{n,n-1}$	$a_{n,n}$	0

4. Viết chương trình hoán vị ma trận thưa

Viết chương trình hoán vị ma trận thưa có m hàng n cột, số liệu lưu trong máy dưới dạng 3 cột q hàng (q là số phần tử khác 0):

m	n	q
...	...	...
i	j	a[i, j]

Hiện lên trên màn hình ma trận lưu kết quả sau khi hoán vị

5. Viết chương trình tra bảng:

Cho bảng số liệu:

	y_1	y_2	....	y	y_i	y_n
x_1	$f(x_1, y_1)$	$f(x_1, y_2)$	...	$f(x_1, y_j)$	...	$f(x_1, y_n)$
x_2	$f(x_2, y_1)$	$f(x_1, y_2)$	...	$f(x_2, y_j)$	...	$f(x_2y_n)$
...	...	...	...	...	...	...
x_i	$f(x_i, y_1)$	$f(x_i, y_2)$	...	$f(x_i, y_j)$	...	$f(x_i, y_n)$
...	...	...	...	...	...	...
x_m	$f(x_m, y_1)$	$f(x_m, y_2)$	...	$f(x_m, y_j)$	...	$f(x_m, y_n)$

Yêu cầu viết chương trình để:

a) Lưu bảng trên vào 1 tệp trên đĩa từ;

b) Khi nhập 1 cặp giá trị (x, y) thì sẽ cho kết quả hàm f(x, y) theo cách tính sau:

+ Nếu x trùng với x_i, y trùng với y_j (i từ 1 đến m, j từ 1 đến n) thì kết quả là:

$$f(x_i, y_j)$$

+ Nếu x trùng với x_i và y nằm trong khoảng từ y_j đến y_{j+1} thì kết quả là:

$$(f(x_i, y_j) + f(x_i, y_{j+1})) / 2$$

+ Nếu y trùng với y_j và x nằm trong khoảng từ x_i đến x_{i+1} thì kết quả là:

$$(f(x_i, y_j) + f(x_{i+1}, y_j))/2$$

+ Nếu x nằm trong khoảng từ x_i đến x_{i+1} và y nằm trong khoảng từ y_j đến y_{j+1} thì kết quả là:

$$(f(x_i, y_j) + f(x_i, y_{j+1}) + (f(x_i, y_j) + f(x_{i+1}, y_j))/4$$

6. Viết chương trình kiểm tra một ma trận vuông đã cho trước có phải là:

+ Ma trận tam giác trên không?

+ Ma trận tam giác dưới không?

+ Ma trận 3 vệt không?

2.5.2. Bài tập về ngăn xếp

1. Dùng kiểu dữ liệu ngăn xếp viết chương trình đổi một số nguyên cơ số 10 sang cơ số 2.

2. Với một số nguyên n >1 cho trước, số nguyên nhỏ nhất d (d >1) chia hết bởi n là 1 số nguyên tố. Ta có thể tìm được các thừa số nguyên tố của n bằng cách trước hết tìm d rồi thay thế n bởi thương số n chia cho d và lặp lại thao tác này cho đến khi n = 1. Dùng kiểu dữ liệu ngăn xếp viết chương trình xác định các thừa số nguyên tố này và hiển thị chúng theo thứ tự giảm dần.

Ví dụ: n = 3960 sẽ phân tích thành $11 \times 5 \times 3 \times 3 \times 2 \times 2 \times 2$.

2.5.3. Bài tập về hàng đợi

Dùng cho các phép toán cơ bản trên hàng đợi ISEMPTYQ, CREATEQ, ADDQ, REMOVEQ để:

1. Lấy ra các phần tử ở cuối hàng đợi nhưng để lại nguyên nội dung hàng đợi;

2. Thêm phần tử vào đầu hàng đợi.

2.5.4. Bài tập về danh sách

Ghép nối, hoàn thiện và chạy thử nghiệm chương trình đối với ví dụ đã cho.

Chương 3
CẤU TRÚC DỮ LIỆU PHI TUYẾN

3.1. BẢN GHI - RECORD

3.1.1. Khái niệm bản ghi

Xét bài toán: viết chương trình quản lý sinh viên, thông tin về một sinh viên gồm có: mã số sinh viên, họ tên, ngày sinh, quê quán, kết quả học tập,... những thông tin này có thể có các kiểu dữ liệu khác nhau: mã số là kiểu nguyên - integer hoặc có thể biểu diễn dưới dạng xâu, chuỗi ký tự - string, họ tên thường là chuỗi ký tự - string, ngày sinh có thể là dạng chuỗi hay dạng số nguyên, quê quán - dạng chuỗi ký tự, kết quả học tập - kiểu số thực - real.... Với các dữ liệu không thuần nhất này không thể tổ chức trong một mảng kiểu array. Để xử lý những thông tin dạng này, trong Pascal dùng một kiểu dữ liệu có cấu trúc là **record** - *bản ghi*. Ta có thể định nghĩa **record** là một phiếu tin về một đối tượng, mỗi **record** chứa một số thành phần dữ liệu đặc trưng cho đối tượng đó, gọi là các trường, mỗi trường đều có một tên và kiểu dữ liệu riêng.

Bản ghi là kiểu dữ liệu phức tạp, không tuyến tính. Bản thân **record** không có tên.

Ví dụ 3.1: Một số đối tượng trong các bài toán quản lý có thể tổ chức dữ liệu theo kiểu bản ghi:

- Hồ sơ nhân sự: họ tên, số nhà, tuổi, nghề nghiệp, nơi công tác, lương,...;
- Địa chỉ: họ tên, số nhà, phố, quận, thành phố, nước;
- Cuốn sách thư viện: tên sách, chuyên ngành, tác giả, năm xuất bản, tập mấy, ký hiệu;
- Sinh viên: họ tên, lớp, khoa, khoá, điểm, học bổng.
- Thiết bị: mã thiết bị, tên thiết bị, nhãn hiệu, năm sản xuất, nước sản xuất, giá, ngày đưa vào sử dụng, nơi sử dụng, tình trạng...

3.1.2. Khai báo kiểu record

Để sử dụng bản ghi trong chương trình, bản ghi phải được khai báo ở phần đầu chương trình. Trong Pascal, bản ghi phải được khai báo kiểu ở phần khai báo *type* với từ khoá *record*, tiếp theo là các tên trường và tên kiểu của chúng, kết thúc bởi từ khoá *end*.

Dạng tổng quát khai báo kiểu *record* là:

```
Type
    R = Record
        <tên trường thứ nhất>:<kiểu dữ liệu trường thứ nhất>;
        ..............
        <tên trường thứ N >:<kiểu dữ liệu trường thứ N >;
end;
```

Ví dụ 3.2: Viết khai báo record cho một số dữ liệu trong ví dụ 3.1.

```
Type
hosons = record
    hoten: string [25];
    tuoi: integer;
    nghe: (moc, ren, xay, co khi);
    donvi: string [30];
    luong: real;
end;
```

```
Type
dia_chi = record
 hoten: string [25];
        sonha: string [10];
        pho: string [10];
        quequan: string [60];
        thanhpho: string [25];
end;
```

Trong những dữ liệu có cấu trúc phức tạp, một bản ghi có thể có nhiều cấp, nghĩa là trường của nó lại là một bản ghi.

Ví dụ 3.3: Bản ghi hồ sơ nhân sự có trường Lịch sử lương, mà bản thân nó lại là một bản ghi chứa các trường Thâm niên, Ngày lên lương cuối cùng, Mức lương hiện nay, Khen thưởng kỷ luật.

Mô tả bản ghi này như sau:

```
Type    luong = record
                    thamnien: integer;
                    ngayll: string[8];
```

```
                    mucluong: real;
                    kyluat: integer;
end;
hosons = record
                    hoten: string [25];
                    tuoi: integer;
                    donvi: string [30];
                    lichslg: luong;
end;
```

Sau khi khai báo kiểu bản ghi bằng Type, trong phần khai báo biến *var* những biến nào có kiểu bản ghi tương ứng thì ta khai báo tên kiểu bản ghi đó. Trong trường hợp tổng quát, khai báo kiểu bản ghi có dạng:

var v1, v2,..., vN: <tên kiểu record>;

Ví dụ 3.4: Hai biến lylich1, lylich2 có kiểu hosons, khi đó ta khai báo:

Var lylich1, lylich2: hosons;

3.1.3. Truy nhập bản ghi

Với bản ghi có thể đọc hay viết giá trị - truy nhập các trường của nó. Việc truy nhập bản ghi được thực hiện bởi cách viết:

<biến kiểu bản ghi>.<tên trường>

Ví dụ 3.5: Tổ chức danh mục chủng loại sản phẩm bê tông theo kiểu *record* như sau:

```
Type
Sanpham = record
                THUTU: integer;
                TENSF: string[15];
                XIMANG: real;
                SAT: real;
                GIASAT: real;
                GIAXIMANG: real;
                ......
end;
```

```
var   COTVUONG, ONGNUOC, PANEL: Sanpham;
Begin        (*Chương trình chính*)
      COTVUONG.XIMANG : = 0.45;
      COTVUONG.GIAXIMANG : = 35500.0;
      COTVUONG.SAT : = 10.72;
      .....
end;
```

Trong ví dụ này ta dùng phép gán := để đưa giá trị vào các trường của *record cotvuong*. Cũng có thể đọc dữ liệu vào từ bàn phím cho bản ghi bởi lệnh *read*.

Ví dụ 3.6: Đọc dữ liệu vào cho các trường của bản ghi trong ví dụ 3.4:

```
write('Ho va ten:'); readln(LYLICH1.HOTEN);
write('Tuoi:'); readln(LYLICH1.TUOI);
.....
write('Muc luong:'); readln(LYLICH1.LICHSLG.MUCLUONG);
.....
```

Để ý là trong lệnh:

```
readln(LYLICH1.LICHSLG.MUCLUONG);
```

Biến MUCLUONG là một trường của bản ghi LICHSLG, mà LICHSLG là một trường của bản ghi LYLICH1.

Trong ví dụ 3.4 hai bản ghi lylich1 và lylich2 có cùng kiểu bản ghi HOSONS, chúng được gọi là các biến **record** trùng kiểu. Các biến **record** trùng kiểu có thể sao chép cho nhau bằng lệnh gán. Đối với hai biến lylich1 và lylich2 ta có thể viết:

```
LYLICH1: = LYLICH2;
```

Khi lệnh này thực hiện, từng trường của **record** LYLICH1 được gán giá trị của trường tương ứng của LYLICH2.

Ngoài phép gán, hai biến **record** cùng kiểu còn có thể so sánh = hoặc <>. Biến bản ghi có thể dùng làm tham số hình thức hay thực sự trong chương trình con. Chúng có thể được tổ chức thành mảng **array**.

Ví dụ 3.7: Khai báo kiểu bản ghi cho bài toán quản lý thiết bị.

```
Type
  MANGTB = record {mảng thiết bị}
        TENVTTB: string[15]; {tên vật t thiết bị}
        NUOCSX: string[15];
        NGAYNHAP: string[8];
        GIATRI: real;
        DONVISD: string[20];
        TINHTRANG: string[15];
end;
var
OTO, QUATDIEN, ONAP, MAYDHOAI: array[1..15] of MANG TB;
```

3.1.4. Câu lệnh with

Truy nhập bản ghi theo cách trên đây phải viết lặp lại nhiều lần tên bản ghi trước mỗi tên trường. Để tránh việc viết lặp lại này, trong PASCAL sử dụng câu lệnh **with** (với), có cú pháp:

```
With<tên biến bản ghi> do
Begin
  <Câu lệnh với tên trường 1>;
  .....
  <Câu lệnh với tên trường thứ n>;
end;
```

Như vậy, tên biến bản ghi chỉ viết một lần sau từ khoá **with**, trong vòng lặp **do ... end** không cần viết lại nó nữa.

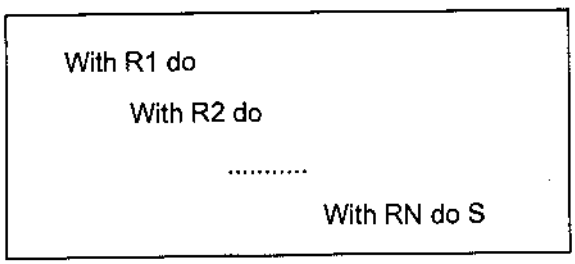
Ví dụ 3.8: Một bản ghi VATTU có các trường TENVT, NUOCSX, NGAYNHP, giatri, donvisd, tinhtrang, ta sẽ đưa dữ liệu vào từ bàn phím như sau:

```
With VATTU do
Begin
      write('Ten vat tu'); readln (TENVT);
      write('nuoc san xuat'); readln (NUOCSX);
```

```
        write('Ngay nhap kho'); readln (NGAYNHP);
        write('Gia tri'); readln (GIATRI);
        write('Don vi su dung'); readln (DONVISD);
        write('Tinh trang'); readln (TINHTRANG);
    end;
```

Đối với bản ghi nhiều thứ bậc:

câu lệnh **with** được tổ chức lồng nhau:



Đơn giản hơn, ta có thể viết lại các lệnh **with** lồng nhau dưới dạng:

with R1, R2,..., Rn **do S;**

3.1.5. Bản ghi có cấu trúc thay đổi

Cấu trúc của bản ghi là các trường với tên, kiểu dữ liệu và kích thước của trường dữ liệu. Những bản ghi vừa xét trên có cấu trúc cố định, không thay đổi. Nếu bản ghi có vài chục trường, mà trong đó chỉ một số trường sử dụng cho một nhóm đối tượng này, một số trường khác cho một nhóm khác, thì tổ chức cố định rất lãng phí bộ nhớ. Chẳng hạn, tổ chức tệp kết quả học tập của sinh viên trong một học kỳ, với nhiều môn học, trong đó:

- Ngành cầu: Mố cầu (MC), Nguyên lý thiết kế cầu (NLTKC), Kết cấu thép (KCT), Tin học ứng dụng (TUD), Tiếng Anh (TA), Thi công cầu (TCC).

- Ngành Đường: Thiết kế đường (TKD), Sức bền vật liệu (SBVL), Trắc địa (TDIA), Địa chất công trình (DIACHAT), Tin học ứng dụng (TUD), Giáo dục quốc phòng 4 (QP4);

- Ngành Xây dựng dân dụng và Công nghiệp: Cơ lý thuyết (CLT), Toán cao cấp (TOAN), Sức bền vật liệu (SBVL), Kết cấu bê tông cốt thép (BTCT), Kỹ thuật thi công (KTTC), Dự toán công trình (DUTOAN), Tin học ứng dụng (TUD);

- Ngành kiến trúc: Vẽ kỹ thuật (VKT), Kiến trúc dân dụng (KTDD), Nguyên lý thiết kế kiến trúc (NLTK), Cấu tạo kiến trúc (CTKT), Toán (TOAN), Triết học (TRIET);

- Ngành Công trình Thuỷ: Cơ học chất lỏng (CCL), Hình học hoạ hình (HHOA), Toán cao cấp (TOAN), Tin học ứng dụng (TUD), Sức bền vật liệu (SBVL), Kỹ thuật thi công (KTTC);

Với cách tổ chức bản ghi có cấu trúc thay đổi hay bản ghi có cấu trúc động, có thể giải quyết được vấn đề phức tạp này.

Bản ghi có cấu trúc thay đổi được chia làm hai phần: phần cố định và phần biến đổi. Phần biến đổi chỉ được là một trường và để ở cuối (tiếng Anh gọi là trường đuôi - **tag fied**), đặt trong từ khoá **case... of**. Tùy theo trường hợp (**case**) trường đuôi nhận giá trị nào thì nó sẽ cấu trúc tương ứng.

Dạng tổng quát của bản ghi cấu trúc thay đổi:

```
Type
<Tên kiểu> = <Kiểu dữ liệu>;
.....
<Tên kiểu bản ghi> = record
                        {phần cấu trúc cố định}
                case <trường thay đổi> : <tên kiểu> of
                <Giá trị của kiểu> : <danh sách của trường>;
                                .....
                        end;
```

Trong đó danh sách trường tuỳ thuộc vào giá trị cụ thể trong danh sách kiểu, dấu ( ) bao danh sách trường là bắt buộc kể cả khi nó rỗng.

Ví dụ 3.9

```
Type
CLASSE = (CAU, DG, TL, KD, MAY, KT);
DSMONHOC = record
```

```
case MONHOC: CLASSE of
        CAU: (MC, NLTKC, KCT, TUD, TA, TCC);
        DG: (TKD, SBVL, TDIA, DIACHAT, TUD, QP4);
        XD: (CLT, TOAN, SBVL, BTCT, KTTC, DUTOAN, TUD);
        KD: (VKT, KTDD, NLTK, CTKT, TOAN, TRIET);
        CTT: (CCL, HHOA, TOAN, TUD, SBVL, KTTC);
end;
```

Trong bản ghi có cấu trúc thay đổi, phần cố định có thể vắng (như trong ví dụ trên) nghĩa là bản ghi chỉ có một trường thay đổi. Bản thân trường thay đổi này lại là một bản ghi mà trong đó có thể chứa một bản ghi thay đổi khác. Bản ghi có cấu trúc thay đổi rất tiện lợi, thay vì một bản ghi có chứa vài chục môn học ta chỉ cần dùng bản ghi biến đổi có số lượng môn học giống như trên thực tế, thường mỗi học kỳ sinh viên học không quá 8 môn.

3.2. KIỂU DỮ LIỆU TỆP

3.2.1. Khái niệm tệp

Tệp là một khái niệm quan trọng của Công nghệ thông tin. Tệp là tập hợp các dữ liệu được tổ chức theo một cách xác định. Cần phân biệt tệp chương trình và tệp dữ liệu. Tệp chương trình là tập hợp các câu lệnh tệp, tạo thành một phần mềm, thực hiện các công việc xử lý thông tin, tệp dữ liệu là tập hợp dữ liệu. Trong giáo trình này tìm hiểu cách thức lưu trữ và xử lý tệp dữ liệu. Tệp dữ liệu có cấu trúc thống nhất là tệp được tạo thành từ các bản ghi có cùng cấu trúc, ngoài ra còn có tệp tạo thành từ các bản ghi không cùng cấu trúc.

Việc tổ chức tệp và cách truy nhập tệp gắn bó với nhau và được gọi chung là phương pháp truy nhập tệp. Chủ yếu có hai phương pháp truy nhập tệp: tuần tự và trực tiếp.

- Truy nhập tuần tự: trong phương pháp này tệp được tổ chức và truy nhập một cách tuần tự, thao tác với tệp phải bắt đầu từ đầu tệp, xong phần tử trước mới đến phần tử sau.

- Truy nhập trực tiếp hay ngẫu nhiên: là phương pháp thao tác với các phần tử tệp theo địa chỉ bất kỳ, không theo tuần tự.

Ví dụ 3.10: Khi xét một danh sách lên lương, theo truy nhập tuần tự thì phải duyệt hết từ đầu đến cuối, theo truy nhập trực tiếp thì ta có thể lúc đầu chỉ duyệt những người đạt một số tiêu chuẩn nào đó, nếu còn chỉ tiêu thì mới xét thêm.

Trong Pascal chỉ xét các tệp tuần tự với các phần tử (bản ghi) cùng kiểu dữ liệu. Trong một số cài đặt cụ thể, khác Pascal chuẩn, xét thêm truy nhập trực tiếp.

Để nhận biết hết tệp khi xử lý tuần tự, trong Pascal có hàm EOF(f) (End of File - kết thúc tệp), *f* là tên biến tệp; hàm này có giá trị *false* khi chưa hết tệp, *true* khi hết tệp. Trong phần này, nói tệp có nghĩa là tệp tuần tự, khi có việc với tệp trực tiếp sẽ nói riêng.

Tuỳ theo cách cấu trúc dữ liệu của tệp mà tệp được phân thành các loại: *tệp mã, tệp văn bản* và tệp *không định kiểu.*

- *Tệp mã*: là tệp mà thông tin của nó được lưu dưới dạng mã 0 và 1. Mỗi phần tử của tệp được tự động đánh số thứ tự từ 0 trở đi. Muốn truy cập vào một phần tử nào đó ta có thể duyệt tuần tự đưa cửa sổ đi qua các phần tử trước đó hoặc chuyển thẳng tới số thứ tự. Nhờ vậy ta có thể sửa chữa nội dung của tệp thuận tiện.

- *Tệp văn bản:* là tệp, dữ liệu được lưu trữ dưới dạng mã ASCII và xếp theo dòng với tín hiệu kết thúc dòng EOLN (End of Line). Độ dài mỗi dòng có thể thay đổi khác nhau. Muốn truy cập vào phần tử nào đó phải tiến hành tuần tự. Muốn ghi hay thêm một phần tử vào tệp ta cần phải đặt cửa sổ vào vị trí cuối tệp.

- *Tệp không định kiểu:* là kiểu tệp đặc biệt được định nghĩa ra trong Turbo Pascal. Trong khi khai báo tệp này người dùng không khai báo kiểu dữ liệu của các thành phần trong tệp.

Tệp được tổ chức theo kiểu nào, sẽ có cách khai báo theo kiểu đó.

3.2.2. Khai báo tệp mã

Tệp sử dụng trong chương trình phải được khai báo. Có khai báo kiểu tệp và khai báo biến tệp.

Khai báo kiểu tệp mã có dạng:

Type

<tên kiểu tệp> = **file of** <kiểu phần tử>;

Trong đó kiểu phần tử có thể là integer, real, char, boolean, kiểu liệt kê, kiểu miền con, kiểu record, array, set (tập hợp). Sau phần khai báo kiểu ta khai báo biến var.

Var

<tên biến tệp> : < tên kiểu tệp>;

Ví dụ 3.11: Xét ví dụ tổ chức ba kiểu tệp: kiểu số nguyên, kiểu số thực và kiểu record. Những tệp số điện thoại là tệp kiểu số nguyên, tệp dữ liệu về lương có kiểu số thực, tệp hồ sơ sinh viên có kiểu bản ghi.

```
Type
        KIEUNGUYEN = file of integer;
        KIEUTHUC = file of real;
        BANGHI = file of record
                HOTEN: string[25];
                LOP: string[10];
                DIEM: real;
                HOCBONG: real;
                        end;
Var
                SODT: KIEUNGUYEN;
                SOLUONG: KIEUTHUC;
                HOSOSV: BANGHI;
```

Chú ý: Sinh viên cần phân biệt tên biến tệp và tên kiểu tệp. Trong ví dụ trên, SODT, SOLUONG, HOSOSV là tên biến tệp, còn KIEUNGUYEN, KIEUTHUC, BANGHI là tên kiểu tệp.

Ví dụ 3.12: Ta có thể tổ chức tệp trong đó các phần tử là mảng array và bản ghi record:

```
Type
        DSLOP = file of record
                                HOTEN = array[1..20] of char;
                                HANHKIEM = char;
                                MON1, MON2, MON3, DTB: real;
                        end (*hết của record*);
```

```
MUCDTB = file of array[1.. 5] of real;
Var
        K51, K52, K53, K54, K55: DSLOP;
        MHOCBONG: MUCDTB;
        TENTEP: string[8];
```

Trong ví dụ này danh sách lớp của các khoá 51, 52, 53, 54, 55 được tổ chức thành các tệp có kiểu dslop, mỗi phần tử là một bản ghi với các trường hoten, hanhkiem, mon1, mon2, mon3, dtb.

3.2.3. Các thao tác với tệp mã

Những thao tác với tệp gồm có:

1. Mở tệp để chuẩn bị ghi dữ liệu lên tệp.
2. Mở tệp để đọc dữ liệu từ tệp.
3. Tiến hành ghi dữ liệu lên tệp.
4. Đọc dữ liệu từ tệp.
5. Đóng tệp. Bao giờ cũng phải đóng tệp ngay sau khi đã xử lý xong phần tử cuối cùng của tệp để bảo vệ dữ liệu.

Khi đã có tệp trên đĩa từ, mở tệp này có ý nghĩa là thiết lập sự làm việc giữa chương trình và bộ nhớ ngoài, nơi lưu trữ tệp, cho phép ta làm việc được với tệp (như sao chép, ghi, xoá,..). Đóng tệp là máy sẽ hoàn tất công việc với tệp, cắt kênh không cho liên hệ với nó nữa. Nếu một tệp chưa có trên đĩa, khi tạo nó lần đầu tiên cũng tương đương với việc mở nó.

3.2.3.1. Thủ tục rewrite - Mở tệp mới để ghi dữ liệu vào

Muốn tạo lập, mở tệp ra ta dùng hai thủ tục Assign và Rewrite.

Cú pháp:

```
ASSIGN (<tên biến tệp>, ' <tên tệp>');
REWRITE (<tên biến tệp>);
```

Trong đó:

+ Tên biến tệp: là tên biến tệp được khai báo trong phần type và var;

+ Tên tệp: tên tệp là tên đặt ở thiết bị nhớ ngoài, tên tệp phải để trong cặp nháy đơn và ghi đầy đủ tên đường dẫn thư mục, trong trường hợp

không có tên đường dẫn thư mục thì được ngầm hiểu là tệp mới tạo sẽ để trong thư mục chứa chương trình chính của phần mềm, đối với Turo Pascal, thư mục đó là C:\TP\BIN.

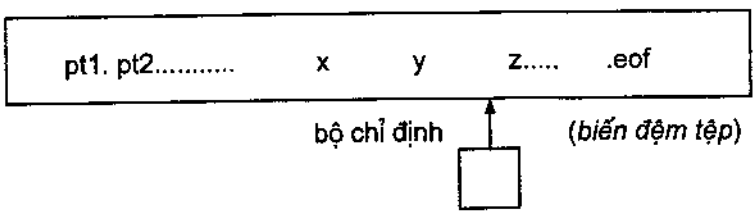
Thủ tục rewrite sẽ chuẩn bị một tệp để viết dữ liệu vào, nếu tệp đã tồn tại thì sẽ được xoá rỗng trừ dấu hiệu kết thúc tệp EOF.

Ví dụ 3.13:

Tạo tệp HOSO.DAT để viết dữ liệu vào ở trên ổ đĩa D:\TP, ta viết:

```
.....
Begin
        write (' Hay go tu ban phim: D:\TP\HOSO.DAT ');
        readln (TENTEP);
        assign (K58, TENTEP);
        rewrite (k58);
End;
```

Khi đã khai báo biến tệp, ta đã ngầm hình thành một biến đệm tệp (file buffer variable) và một bộ chỉ định (indicator) ứng với nó. Biến đệm tệp có kiểu cùng kiểu với kiểu phần tử của tệp. Tại mỗi thời điểm chỉ có thể làm việc với một phần tử của tệp, thông qua biến đệm này, bộ chỉ định có chức năng chỉ ra vị trí của phần tử tệp. Chúng ta có thể hình dung một tệp T có dãy các phần tử x, y, z,... với các biến đệm tệp và bộ chỉ định như sau:



Kết hợp hai chức năng nhớ đệm và chỉ vị trí, người ta ký hiệu biến đệm là:

<biến đệm tệp> = <tên biến tệp> ↑

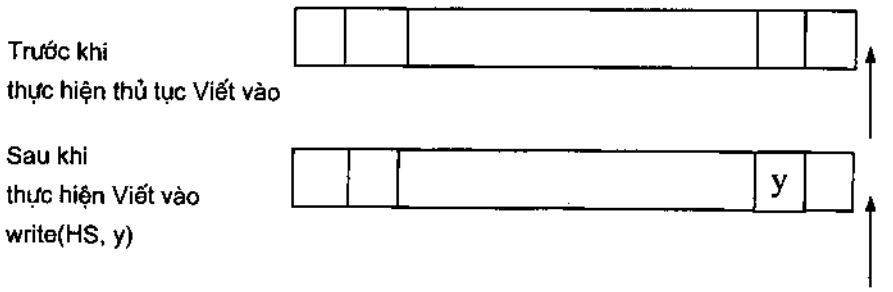
Chẳng hạn, biến tệp là H1 thì biến đệm tương ứng là H1↑, nếu biến tệp là HS thì biến đệm là HS↑. Khi thủ tục rewrite bắt đầu làm việc, xoá rỗng tệp sẽ chỉ vào ký hiệu kết thúc tệp EOF.

3.2.3.2. Thủ tục write - ghi (viết) dữ liệu vào tệp

- *Cú pháp:* Write (<tên biến tệp>, x);

- *Chức năng:* Viết giá trị biểu thức x vào tệp đã được mở với <tên biến tệp>.

Ví dụ 3.14: Một tệp HS trước và sau khi thực hiện thủ tục write có thể hình dung như sau:



Ví dụ 3.15: Cho tên tệp GTGT.DAT, kiểu integer. Viết các giá trị n! với n từ 1 đến 15 cho các phần tử tệp.

```
Program giatriGT;
Var
      i, v: integer;
      T: file of integer;
Begin
            assign (T, 'GTGT.DAT');
            rewrite (t);
            V: = 1;
            for I: = 1 to 15 do
                        begin V: = V*i;
                              write (T, V)
                        end;
   close (T);
End.
```

3.2.3.3. Đóng tệp

- *Cú pháp:* Close (<tên biến tệp>);

- *Chức năng:* đóng tệp đã làm việc xong, ở ví dụ trên đã sử dụng lệnh này đóng tệp T.

3.2.3.4. Mở tệp đã tồn tại - reset

- *Cú pháp:*

ASSIGN (<tên biến tệp>, '<tên tệp>');

RESET (<tên biến tệp>);

Khi tệp đã tồn tại, thủ tục reset sẽ thiết lập trạng thái sẵn sàng làm việc với nó, đưa vị trí đọc về đầu tệp. Ta có thể hình dung tệp T sau khi mở bởi reset:

Ví dụ 3.16:

```
assign(t, 'GTGT.DAT');
reset(t);
```

Giải thích ví dụ: Mở tệp giai thừa GTGT.DAT để trong thư mục hiện thời của Turbo Pascal.

3.2.3.5. Thủ tục đọc dữ liệu từ tệp - read

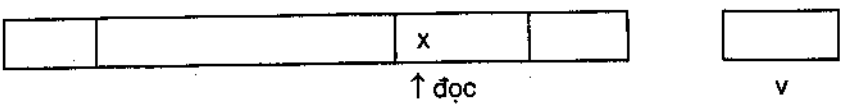
- *Cú pháp:*

READ (<tên biến tệp>, v);

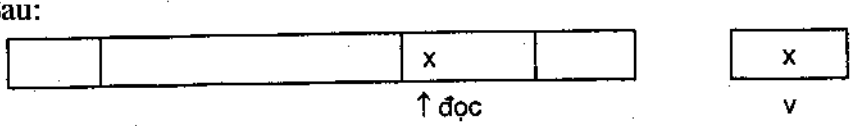
- *Chức năng:* Sao chép giá trị phần tử hiện thời của tệp cho biến v, sau đó đưa vị trí đọc dịch sang phần tử kế tiếp. Kiểu phần tử của tệp phải trùng với kiểu của biến v.

Ta có thể hình dung tệp T trước và sau khi thực hiện read như sau:

Trước:



Sau:



Khi biến v là một danh sách, việc đọc tệp sẽ tiến hành cho đến khi hết tệp hoặc hết danh sách. Dùng điều kiện kiểm tra kết thúc tệp eof <tên biến tệp> ở đây là cần thiết. Các phần tử của tệp được đọc một cách tuần tự, hết phần tử trước mới đến phần tử sau.

Ví dụ 3.17:

```
program TICH;
Var
        TEPNGUYEN : file of integer;
            i, j: integer;
Begin
   Assign (TEPNGUYEN, 'TICHSN.DAT');
   reset (TEPNGUYEN);
   I: = 1;
   while not (eof(TEPNGUYEN) and MAXINT > I do
            Begin
                read (TEPNGUYEN, J);
                I: = I*J
            end;
   writeln ('TICH = ', I);
   Close (TEPNGUYEN);
   readln
end.
```

Trong ví dụ này dùng điều kiện (MAXINT > 1) để dừng tính toán khi i vượt quá khả năng lưu trữ số nguyên trong máy.

Người ta phân biệt các tệp ngoài và tệp trong. Tệp ngoài tồn tại độc lập, không phụ thuộc vào chương trình. Chúng có thể được đưa vào chương trình với tư cách là tham số ở đầu chương trình. Thông thường tệp ngoài để ở đĩa mềm hay đĩa cứng. Tệp được sinh ra và sử dụng trong quá trình chương trình làm việc gọi là tệp trong. Tên của nó có thể xuất hiện ở đầu chương trình nếu nó có liên hệ với thiết bị ngoại vi.

Khi chương trình kết thúc, các tệp trong cũng bị xoá vì vậy chúng còn được gọi là các tệp tạm thời (temporary file) hay các tệp vết (scratch). Thông thường các tệp trong được sử dụng trong các chương trình con procedure hay function.

Đối với các tệp tuần tự trong PASCAL, trong cùng một thời điểm không thể vừa đọc ra vừa viết vào, vì vậy khi xử lý, tối thiểu cần phải tạo ra hai tệp, một để đọc ra, một để viết vào. Ở những thời điểm chạy chương trình khác nhau, có thể thay đổi vai trò của hai tệp này cho nhau.

3.2.3.6. Một số thủ tục xử lý tệp mã

+ Tìm vị trí của phần tử tệp.

Cú pháp: seek (<tên biến tệp>, N);

Trong đó N là số nguyên, chỉ phần tử ở vị trí N và đưa con trỏ biến đệm tệp về đó để cập nhật.

Bằng lệnh seek, ta có thể truy nhập không tuần tự vào một tệp tuần tự.

+ Đếm số phần tử của tệp.

Cú pháp: filesize(<tên biến tệp>);

Kết quả sẽ cho một số nguyên dương, khi tệp rỗng sẽ cho kết quả 0.

+ Xác định vị trí hiện thời của biến đệm tệp.

Cú pháp: filepos(<tên biến tệp>);

Kết quả sẽ cho một số nguyên dương, khi tệp rỗng sẽ cho kết quả là 0.

+ Xoá tệp trên đĩa từ.

Cú pháp: Erase(<tên biến tệp>);

+ Đổi tên tệp

Cú pháp: Rename(<tên biến tệp>, <tên tệp mới. Kiểu tệp>);

Thủ tục này sẽ đổi tên tệp cũ tương ứng với **tên biến tệp** thành tên mới là **<tên tệp mới. kiểu>.** <Tên tệp mới. kiểu> không được trùng với tên tệp đã có trong thư mục đang làm việc.

+ Hàm xoá bỏ chế độ không kiểm tra lỗi khi làm việc với tệp

Cú pháp: IORESULT

Hàm này không có tham số, kết quả cho một giá trị nguyên bằng 0.

3.2.4. Tệp văn bản - TEXT

3.2.4.1. Khái niệm

Tệp văn bản là tệp đặc biệt trong Pascal, nó chứa các bản ghi có thể có độ dài khác nhau, mỗi bản ghi là một dòng, mỗi dòng có ký tự kết thúc là EOL.

Ví dụ 3.18:

Nội dung một tệp văn bản đơn giản:

HOANG BAC **eol**
AN **eol**
02/10/75 **eol**
HA NOI

3.2.4.2. Khai báo tệp văn bản

Cú pháp: Type <tên kiểu tệp> = Text;
Var <tên biến tệp>: <tên kiểu tệp>;

Hoặc khai báo trực tiếp:

Var <tên biến tệp> : Text;

Ví dụ 3.19:

a) Type NS = Text;
SL = Text;
Var Nhan_su: NS;
SoLuong: SL;

b) Var Nhan_su, SoLuong: Text;

3.2.4.3. Các thao tác với tệp văn bản

Những thao tác với tệp Text là write, writeln, read, readln, eoln. Những thủ tục này có thể làm việc với các kiểu dữ liệu đơn giản như char, string, integer, real, boolean.

** Ghi dữ liệu vào tệp văn bản*

Sau khi mở tệp để ghi ra, ta có thể tiến hành ghi dữ liệu lên tệp, mỗi phần tử tệp là một dòng, mỗi dòng có ký hiệu EOL để ngăn cách.

Cú pháp câu lệnh đối với tệp văn bản có phần khác với tệp mã:

1. WRITE (<biến tệp>, <danh sách biểu thức>);

2. WRITELN (<biến tệp>, <danh sách biểu thức>);

3. WRITELN (<biến tệp>);

+ Thủ tục **WRITE (<biến tệp>, <danh sách biểu thức>);** máy sẽ xác định giá trị biểu thức, rồi ghi vào tệp theo trình tự từ trái qua phải.

+ Thủ tục **WRITELN (<biến tệp>, <danh sách biểu thức>);** cũng hoạt động giống như Write nhưng sau khi ghi xong giá trị của biểu thức cuối cùng máy đưa thêm dấu hiệu hết dòng (EOL) vào tệp.

+ Thủ tục **WRITELN (<biến tệp>);** Chèn thêm dấu hiệu hết dòng vào tệp. Dấu hiệu hết dòng là một cặp ký tự điều khiển CR (Catriage Return - quay về đầu dòng) và LF (Line Feed - nhảy dòng).

* *Đọc dữ liệu từ tệp văn bản*

- *Cú pháp:*

1. READ (<tên biến tệp>, <danh sách biến>);

2. READLN (<tên biến tệp>, <danh sách biến>);

3. READLN (<tên biến tệp>);

Ở đây danh sách biến có thể có nhiều biến, chúng được ngăn cách nhau bằng dấu phẩy. Kiểu dữ liệu của các biến không nhất thiết phải giống nhau.

+ Thủ tục **READ (<tên biến tệp>, <danh sách biến>);** sẽ đọc các giá trị của tệp, rồi gán giá trị đọc được cho các biến đã chỉ ra trong câu lệnh, đọc xong không xuống dòng.

+ Thủ tục **READLN (<tên biến tệp>, <danh sách biến>);** sẽ đọc các giá trị của tệp, rồi gán kết quả đọc được cho biến trong câu lệnh. Sau khi đọc xong giá trị cho biến cuối cùng, cửa sổ tệp tự động chuyển đến đầu dòng dưới, bỏ qua mọi giá trị còn thừa của dòng đến EOL.

+ Thủ tục **READLN (<tên biến tệp>);** đưa cửa sổ tệp sang đầu dòng tiếp theo mà không đọc gì cả.

* *Hàm kiểm tra kết thúc dòng*

Cú pháp: eoln(<tên biến tệp>);

Hàm này cho kết quả false khi chưa kết thúc dòng.

Chú ý:

- Với tệp văn bản không thể dùng thủ tục SEEK hoặc hàm FILESIZE hoặc hàm FILEPOS để tìm vị trí phần tử, để đếm kích thước tệp, để xác định vị trí biến đệm tệp. Song, ta có thể dùng các hàm sau:

+ SEEKEOLN (<biến tệp>): Xác định xem cửa sổ tệp có ở vào vị trí cuối dòng không. Giá trị của hàm là true hoặc false. Trước khi kiểm tra EOLN nó nhảy qua các dấu cách và dấu Tab.

+ SEEKEOF (<biến tệp>): Xác định xem cửa sổ tệp có ở vào vị trí cuối tệp không. Giá trị của hàm là true hoặc false. Trước khi kiểm tra EOF nó nhảy qua các dấu cách và dấu Tab và dấu cách dòng.

Ví dụ 3.20:

```
Program Filetext;
    Uses crt, dos;
    Var tepgoc, tepkq: Text;
        Bien, tiep: char;
    V, tentep, tepmoi: string[12];
Begin
Clrscr;
Write('Nhap ten tep goc:');Readln(tentep);
Assign(tepgoc, tentep);
Rewrite(tepgoc);
Repeat
    Write('Nhap thong tin vao tep goc:');Readln(v);
    Writeln(tepgoc, v);
    Write('Co Nhap tiep khong(c/k):');Readln(tiep);
Until upcase(tiep) = 'K';
Close(tepgoc);
Write('Nhap ten tep goc:');Readln(tentep);
Assign(tepgoc, tentep);
Reset(tepgoc);
```

```
While not eof(tepgoc) do
    Begin
        While not eoln(tepgoc) do
            Begin
                Read(tepgoc, bien);
                Writeln(bien:12);
            End;
    End;
Close(tepgoc);
Readln
End.
```

Ví dụ 3.21:

Sử dụng bản ghi và tệp chương trình quản lý vật tư:

```
Program quanlyvattu;
Type vttb = record
        Matb:integer;
        Tentb:string[20];
        Nuocsx:string[15];
        Namsx:integer;
        Giatri:real;
        Soluong:integer;
        Noisd:string[25];
        Tinhtrang:string[10];
    End;
Mang = array[1..100] of vttb;
Var
I, j, n, m :integer;
Bfile:text;
Hstb:mang;
Matim:integer;
```

```
    Ten:string]15];
    Procedure nhap;
    Begin
    Write('Nhap so loai thiet bi : ');readln(n);
    For i: = 1 to n do
    With hstb[i] do
            Begin
                    Write('Ma thiet bi thu ', I, ' : ');readln(matb);
                    Write('Ten thiet bi thu ', I, ' : ');readln(tentb);
                    Write('Nuoc san xuat : ');readln(nsxb);
                    Write('Gia tri : ');readln(giatri);
    Write('So luong thiet bi : ');readln(soluong);
    Write('Noi su dung : ');readln(nsd);
    Write('Tinh trang : ');readln(tinhtrang);
            End;
    Assign(bfile, 'hstb.dat');
    Rewrite(bfile);
    Writeln(bfile, n);
    For i: = 1 to n do with hstb[i] do
        Begin
                Writeln(bfile, matb);
                Writeln(bfile, tentb);
                Writeln(bfile, nsx);
                Writeln(bfile, giatri);
                Writeln(bfile, soluong);
                Writeln(bfile, noisd);
                Writeln(bfile, tinhtrang);
        End;
Close(bfile);
End;
```

```
Procedure ds_vttb;
Begin
        Assign(bfile, 'hstb.dat');
        Reset(bfile);
        Writeln(bfile, n);
        Writeln(n);
For i: = 1 to n do with hstb[i] do
Begin
        Read(bfile, matb);
        Read(bfile, tentb);
        Read(bfile, nsx);
        Read(bfile, giatri);
        Read(bfile, soluong);
        Read(bfile, noisd);
        Read(bfile, tinhtrang);
        Writeln(matb, tentb, nsx, giatri, soluong, noisd, tinhtrang);
End;
Close(bfile);
Procedure timkiem;
      Begin
          Write('Nhap ma thiet bi can tim :');readln(matb);
          For i: = 1 to n do
                If hstb[i].matb = matim then
                With hstb[i] do
                    Begin
                          Writeln('ma thiet bi :', matb);
                          Writeln('Ten thiet bi: ', tentb);
                          Writeln('Nuoc san xuat :', nsx);
                          Writeln('Gia tri :', giatri);
                          Writeln('So luong :', soluong);
```

```
                    Writeln('Noi su dung :', noisd);
                    Writeln('Tinh trang :', tinhtrang);
                End;
        Readln
        End;
Begin {chương trình chính}
        Nhap;
        Ds_vttb;
        Timkiem
End.
```

3.3. KIỂU DỮ LIỆU TẬP HỢP

3.3.1. Tập hợp và khai báo kiểu tập hợp

Trong toán học ta đã biết tập hợp gồm những phần tử có cùng một số tính chất nào đó, ví dụ tập các số tự nhiên, tập các số thực, tập các số nguyên, tập các chữ cái. Một tập không có phần tử nào gọi là tập trống, ký hiệu là $\varnothing$.

Trong Công nghệ thông tin sử dụng tập hợp như là một loại dữ liệu có cấu trúc, được hợp thành từ những phần tử có cùng kiểu dữ liệu vô hướng (integer, char, boolean, liệt kê, byte, miền con) trừ kiểu thực **real** không đếm được. Số lượng phần tử cực đại trong tập hợp tuỳ thuộc vào từng cài đặt cụ thể, ví dụ Turbo Pascal con số đó là 256.

Khi sử dụng dữ liệu kiểu tập hợp thì phải định nghĩa kiểu tập hợp cho chúng. Kiểu dữ liệu vô hướng của các phần tử của nó phải được xác định qua khai báo **type** kiểu dữ liệu tập hợp hoặc khai báo trực tiếp trong phần **var**.

3.3.1.1. Khai báo kiểu tập hợp

Cú pháp:

```
Type
        <Tên kiểu tập hợp> = set of <kiểu vô hướng>;
```

Khai báo này đã gán cho tên kiểu tập hợp một miền giá trị xác định - là một tập hợp các phần tử, mỗi phần tử có kiểu dữ liệu đã viết trong câu lệnh. Tên kiểu tập hợp này sẽ được dùng trong khai báo biến **var**. Cũng như các kiểu dữ liệu khác, trong khai báo biến **var** có thể có khai báo trực tiếp kiểu dữ liệu cho tên biến kiểu tập hợp.

3.3.1.2. Ví dụ

Ví dụ 3.22: Một số khai báo kiểu tập hợp.

```
Type
    MAU = (BLUE, BLACK, RED, WHITE); {các màu sắc}
    NGAY = (HAI, BA, TU, NAM, SAU, BAY); {thứ trong tuần}
    NGUYEN = set of 0..256; {tập các số nguyên}
Var
    P, Q: NGUYEN;
    WORKDAY: set of NGAY; {tập các ngày làm việc}
    COLOR: set of MAU; {tập các màu sắc}
    CHUSO: set of 0..9; {tập các chữ số}
```

Biểu diễn tập hợp này thiết lập một tập hợp trong PASCAL được thực hiện bằng cách biểu diễn liệt kê hay biểu diễn miền con, đặt trong cặp ngoặc vuông (đừng nhầm với dấu ngoặc vuông trong phần DOS).

Ví dụ 3.23:

[ ] (tập trống), [...256] (tập các số nguyên từ 0, 1, ... 256)

[true, false] (tập giá trị logic)

['i', 'j'...'n', 'y', 'x'] (tập hợp các chữ từ i đến n và hai chữ y, x)

[XANH, DO, TIM, VANG, TRANG, DEN] (tập các màu).

3.3.2. Các phép toán đối với dữ liệu kiểu tập hợp

Đối với các dữ liệu kiểu tập hợp có các phép toán gán tập hợp, hợp (+), giao (*), hiệu (-) hai tập hợp và các phép tính quan hệ, phép thuộc về (in). Ta lần lượt xét chúng.

3.3.2.1. Phép gán

Cú pháp: X: = E;

Trong đó X là biến kiểu tập hợp, E.là biểu thức tập hợp, các phần tử của E và tập X phải cùng kiểu cơ sở.

Ví dụ 3.24:

Cho P, Q là biến tập hợp kiểu nguyên từ 0 đến 256 trong ví dụ 3.22, khi đó có thể gán

P: = [18..60]; Q: = [50];

Đối với các biến khác:

WORKDAY: = [HAI, BA]; COLOR: = [BLACK];

Tập rỗng có thể gán cho mọi biến tập có kiểu vô hướng bất kỳ.

*3.3.2.2. Các phép hợp, tuyển (giao), hiệu (+, *, -)*

Ba phép này giống như trong lý thuyết tập hợp. Nếu A và B là hai tập hợp cùng kiểu thì:

A + B là một tập hợp gồm những phần tử thuộc A hoặc thuộc B;

A * B là một tập hợp gồm những phần tử thuộc đồng thời A và B;

A - B là một tập hợp gồm những phần tử thuộc A nhưng không thuộc B.

Ví dụ 3.25:

Cho A: = [1, 3, 5, 7, 9, 10]; B: = [2, 4, 6, 8, 10, 20];

Khi đó A + B sẽ là [1..20]; A * B sẽ là [10]; A - B là [1, 3, 5, 7, 9].

3.3.2.3. Các phép tính quan hệ

Các phép toán quan hệ đối với các tập hợp hay biểu thức tập hợp cho kết quả kiểu boolean. Nếu E là biểu thức tập hợp, A và B là hai tập cùng kiểu thì:

E **in** A cho true nếu E thuộc A, ngược lại là false;

A = B cho true nếu A trùng (bằng) B, ngược lại false;

A<>B cho true nếu hai tập A và B khác nhau, ngược lại false;

A< = B cho true nếu A là tập con của B, ngược lại false;

A> = B cho true nếu B là tập con của A, ngược lại false.

Ví dụ 3.26: Viết chương trình tìm tất cả các số nguyên tố của 100 số tự nhiên đầu tiên.

```
Program SNTO;
Const N = 100;
Type THOP = set of 2..N.
Var
    N1, NEXT: integer;
    SNTO1, STN: THOP;
Begin
    STN: = [2..N]; SNTO1: = [1]; NEXT: = 2;
While STN <>[ ] do
    Begin N1: = NEXT;
        While N1 < = N do
            Begin
                STN: = STN - [N1]; N1: = N1 + NEXT
            end;
SNTO1: = SNTO1 + [NEXT];
Repeat NEXT : = NEXT + 1
Until (NEXT in STN) or (NEXT>N)
End, (Viết số nguyên tố*)
for N1: = 1 to N do
 if N1 in SNTO1 then write (N1);
writeln
end.
```

Sử dụng kiểu dữ liệu tập hợp rất tiện lợi khi diễn đạt các mệnh đề tính toán. Trong ví dụ sau đây, thay vì một loạt phép tính kiểm tra xem ký tự đưa vào từ bàn phím có phải là chữ cái không:

```
If (KT = 'A') or (KT = 'B') or.... or (KT = 'Z') then <câu lệnh>;
```

Ta chỉ cần viết:

```
If KT in 'A'..'Z' then <câu lệnh>;
```

Ví dụ 3.27:

```
Program nhapDat;
Var
KT : char; Error: boolean;
X: set of 'A'..'Z';
Begin write ('Dua vao ky tu:'); readln (KT);
Repeat Error: = false;
If not (KT in X) then Error: = true;
Write ('Dua vao ky tu:'); readln (KT);
While KT <> ' ' do
Begin
if not (KT in ['A'..'Z', '0'..'9']) Then Error: = true;
Write ('Dua vao ky tu:'); readln (KT);
End;
Writeln;
If Error then writeln ('Vao sai');
Write ('Nhap vao ky tu:'); readln (KT)
Until KT = '.'
End.
```

3.4. CON TRỎ VÀ CẤU TRÚC DỮ LIỆU ĐỘNG

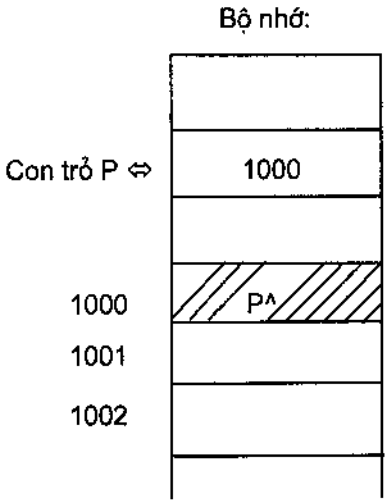
3.4.1. Khái niệm

Dùng con trỏ với các thủ tục new, dispose sẽ cho khả năng cấp phát và giải phóng các vùng nhớ cho các biến trong khi thực hiện chương trình, không cần giới hạn kích thước của vùng lưu trữ. Những dữ liệu được xác định trước bởi khai báo tên trong chương trình và tồn tại trong suốt quá trình chương trình thực hiện gọi là ***dữ liệu tĩnh***. Dữ liệu mà có thể được sinh ra và mất đi trong quá trình thực hiện chương trình gọi là ***dữ liệu động***. Việc truy nhập dữ liệu động phải thông qua một biến con trỏ, trỏ tới bộ nhớ của biến động (dynamic variable). Ở mỗi thời điểm tính toán, sẽ có một vùng nhớ nào đó liên kết với biến động này, ngược lại đối với

biến tĩnh, vùng nhớ liên kết với nó được cấp phát khi dịch chương trình và cố định trong quá trình thực hiện chương trình.

Ký hiệu p là biến kiểu con trỏ, p^ sẽ là biến động, chứa giá trị dữ liệu ở vùng nhớ mà con trỏ p chỉ đến.

Chúng ta có thể mô tả con trỏ như sau:



Trong sơ đồ trên, con trỏ P có giá trị là 1000 - là địa chỉ của ô nhớ mà chứa nội dung của biến động P^.

Trong thực tế chúng ta ít quan tâm đến địa chỉ thực của con trỏ P (*trong ví dụ trên, là địa chỉ ô nhớ có giá trị 1000*) mà chỉ là nội dung của nó, tức là địa chỉ của biến động P^ (*là giá trị 1000*).

Khi con trỏ chưa được định nghĩa hay rỗng thì biến động P^ là chưa được xác định.

Đối với con trỏ có thể thực hiện các phép gán và các phép so sánh =, < >.

Phép so sánh =, < > dùng để so sánh hai con trỏ xem chúng có trỏ đến cùng một địa chỉ hay không hoặc chúng có rỗng hay không.

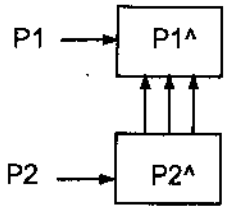
Các phép so sánh sau là hợp lệ:

P1 = P2;

P1< > Nil;

- Đối với biến động P^ Phép gán là gán nội dung dữ liệu:

P1^ : = P2^;



Việc tạo ra biến động cũng như xoá nó đi để giải phóng bộ nhớ được tiến hành nhờ các thủ tục NEW và DISPOSE đã có sẵn của Turbo Pascal. Để truy nhập vào các biến động, được tiến hành nhờ có các biến con trỏ (Pointer Variable). Các biến con trỏ được định nghĩa giống như các biến tĩnh trong phần khai báo ở đầu chương trình. Nó sẽ được dùng để chứa địa chỉ của các biến động.

3.4.2. Khai báo con trỏ, biến động

3.4.2.1. Khai báo con trỏ

Kiểu con trỏ là một kiểu trỏ đến (chỉ đến) một vùng chứa dữ liệu của các biến động, các biến này có thể có các kiểu dữ liệu như đã học như Integer, Real, Char, Boolean, String, Array, Record... Để phân biệt với kiểu dữ liệu tĩnh, kiểu con trỏ có dấu ^ đứng trước. Ví dụ: ^Integer, ^Real, ^Char, ^Boolean, ^String, ^Array, ^Record...

Giống như các kiểu dữ liệu đã xét, trong chương trình nếu có sử dụng kiểu con trỏ, nó phải được khai báo như các kiểu dữ liệu khác bằng lệnh TYPE như sau:

TYPE <tên kiểu con trỏ> = ^<kiểu dữ liệu >;

VAR <danh sách biến con trỏ>: <tên kiểu con trỏ>;

Trong đó:

- Tên kiểu con trỏ: là tên được đặt theo qui tắc đặt tên của ngôn ngữ Pascal.

- Kiểu dữ liệu: là một trong các kiểu dữ liệu như Integer, Real, Char, Boolean, Record...

- Dấu ^ : đứng trước kiểu dữ liệu để chỉ đó là kiểu con trỏ.

Ví dụ: Cần khai báo 4 kiểu con trỏ có tên là IntPtr, CharPtr, RecPtr và DcPtr:

```
Type   IntPtr = ^Integer;
       CharPtr = ^Char;
       RecPtr = ^Record
              Hoten : String[25];
              HSLuong:Real;
              NamCongtac:Integer
              End;
       DcPtr = ^DiaChi; {Kiểu DiaChi được khai báo sau}
       DiaChi = Record
       SoNha: String[10];
       DuongPho: String[25];
       ThanhPho: String[15]
    End;
Var a, b: IntPtr;
    c, d: CharPtr;
    e, f: RecPtr;
    g, h: DcPtr;
```

Chú ý: Turbo Pascal cho phép khai báo kiểu con trỏ trước khi khai báo kiểu của dữ liệu hoặc kiểu của biến động (Xem ví dụ trên DcPtr).

** Có thể khai báo trực tiếp:*

```
Var    a, b: ^Integer;
       C, d: ^Char;
       E, f: ^Record
  Hoten : String[25];
  HSLuong:Real;
  NamCongtac:Integer
      End;
  G, h: ^DiaChi; {Con trỏ khai báo trước khai báo DiaChi }
  DiaChi = Record
```

SoNha: String[10];
DuongPho: String[25];
ThanhPho: String[15]
End;

3.4.2.2. Biến động và truy nhập biến động

Cho p là biến kiểu con trỏ, p^ sẽ là biến động. Để truy cập vào một biến động có địa chỉ nằm trong biến con trỏ ta dùng dấu ^ đứng liền sau tên của biến con trỏ. Ví dụ: a^, b^, c^, d^, e^, f^, g^ và h^.

Thông qua khai báo, chương trình dịch sẽ cấp phát vùng nhớ cho các biến con trỏ để truy nhập vào biến động ở ví dụ trên như sau:

- Biến con trỏ a, b chứa địa chỉ của biến động a^, b^; là các vùng nhớ chứa số nguyên.

- Biến con trỏ c, d sẽ chứa địa chỉ của biến động c^, d^; là các vùng nhớ chứa ký tự.

- Biến con trỏ e, f sẽ chứa địa chỉ của biến động e^, f^; là các vùng nhớ chứa dữ liệu kiểu bản ghi.

- Biến con trỏ g, h sẽ chứa địa chỉ của biến động g^, h^. Đó là các vùng nhớ chứa dữ liệu kiểu bản ghi.

** Chú ý:* Để truy nhập vào biến con trỏ kiểu mảng ta phải viết dấu ^ sau tên biến mảng nhưng trước chỉ số, ví dụ: a^[i]. Ngược lại, để truy nhập đến mảng các con trỏ ta phải viết dấu ^ sau chỉ số, ví dụ: b[i]^. (Xem ví dụ phần sau).

3.4.2.3. Cấp phát vùng nhớ cho con trỏ

Trước khi muốn truy cập đến một biến động, ta phải tiến hành cấp phát vùng nhớ cho con trỏ của nó bằng thủ tục NEW như sau:

- *Cú pháp:* **NEW(p)**; Với p là con trỏ.

- *Tác dụng:* Nhờ có lệnh trên, chương trình sẽ tạo ra một vùng nhớ có kiểu và kích thước tuỳ theo kiểu và kích thước đã khai báo của p. Sau đó ta mới có thể truy cập vào biến động bằng cách kèm theo dấu ^ sau biến. Nếu trong một chương trình ta dùng n lần New(p) liên tiếp, thì máy sẽ tạo ra n con trỏ có cùng một kiểu. Khi đó, con trỏ p luôn trỏ tới biến động p^.

- *Ví dụ:* Xem ví dụ ở phần dưới.

3.4.2.4. Giải phóng vùng nhớ của con trỏ

- *Tác dụng:* Sau khi làm việc xong, một con trỏ không được dùng tới nữa, ta có thể giải phóng nó để thu hồi lại ô nhớ nó chiếm dụng dành cho công việc khác. Muốn vậy ta phải dùng thủ tục DISPOSE như sau:

- *Cú pháp:* **DISPOSE(p)**; với p là con trỏ.

- *Ví dụ:* Xem ví dụ ở phần dưới.

3.4.3. Các thao tác đối với biến con trỏ

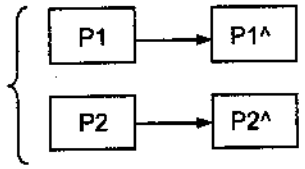
3.4.3.1. Phép gán (: =)

- *Ví dụ:*

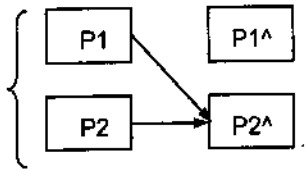
```
Var   p1, p2 : ^Integer;
Begin
        ...
      P1: = P2;
```

Lệnh gán này sẽ làm cho hai con trỏ P1 và P2 cùng chỉ đến một vùng chứa dữ liệu:

Trước khi gán:



Sau khi gán:



3.4.3.2. So sánh hai biến con trỏ cùng kiểu

Ta chỉ có thể so sánh hai biến con trỏ cùng kiểu bằng các phép so sánh = (bằng) và <> (khác).

- *Ví dụ:* **if** a = b **then** Writeln('a bằng b');

3.4.3.3. Hằng con trỏ NIL

Nil là một giá trị hằng đặc biệt dành cho biến con trỏ để báo con trỏ không chỉ vào đâu cả. Nếu ta gán hằng Nil này cho một biến con trỏ thì có nghĩa biến đó không chỉ vào đâu cả.

- *Ví dụ:* a: = Nil; { a không chỉ vào địa chỉ nào }

3.4.3.4. Chú ý:

- Các giá trị của biến con trỏ không thể đọc từ bàn phím bằng thủ tục Read và Readln cũng như không thể đưa ra màn hình hoặc máy in bằng thủ tục Write và Writeln.

- *Ví dụ:* Cần nhập và in các giá trị cho biến con trỏ kiểu bản ghi.

```
Program ban_ghi_dong;
  uses crt;
  Type kbg = Record
       Ma_CB:Integer;
       ht:string[25];
       DonVi:string[5];
       HesoLg:real
     End;
  Var  HsCb:array[1..20] of ^kbg;
    i,n:integer;
  Begin
    Clrscr;    Write('N= ');Readln(n);
    Writeln('Nhap so lieu vao mang dong:');
    For i:=1 to n do
    Begin
       New(HsCb[i]);
       Write('Ma_CB[',i,']= ');Readln(HsCb[i]^.Ma_CB);
       Write('Ho ten = ');Readln(HsCb[i]^.ht);
       Write('Don Vi ');Readln(HsCb[i]^.DonVi);
       Write(' HesoLg = ');Readln(HsCb[i]^.HesoLg);
       Dispose(HsCb[i]);
    end;
    Writeln('Dua du lieu mang dong ra man hinh:');
    Writeln('-------------------------------------------------------');
    Writeln(': Ma_CB :    Ho ten          : DonVi :He so Luong:');
    Writeln(':----------:--------------------:--------:--------------:');
```

```
For i:=1 to n do
Begin
   Write(HsCb[i]^.Ma_CB:4);
   Write(HsCb[i]^.ht:25);
   Write(HsCb[i]^.DonVi:7);
   Write(HsCb[i]^.HesoLg:8:1);
   Writeln;
end;
Writeln(':-----:----------:-----:----------:');
Readln;
End.
```

3.4.4. Một số vấn đề mở rộng, đi sâu đối với con trỏ

3.4.4.1. Những kiến thức nâng cao về con trỏ

- Cần phân biệt sự khác nhau giữa biến thông thường và biến kiểu con trỏ.

Trong kỹ thuật lập trình, con trỏ là khái niệm tương đối trừu tượng và khó hiểu đối với sinh viên. Thông thường sinh viên chỉ quen với các tên biến cụ thể, khi viết c:= a+b thì họ hình dung a, b là 2 giá trị cụ thể và sẽ cho c một giá trị cụ thể, là tổng của 2 số a, b. Khi a và b là 2 biến con trỏ thì a trỏ về địa chỉ chứa giá trị a^; b trỏ về địa chỉ chứa giá trị b^ nên phép cộng a+b không hợp lệ nữa (không được cộng 2 con trỏ). Phép cộng a^+b^ sẽ là tổng của 2 số ở tại địa chỉ mà a và b trỏ đến.

- Giá trị thực sự của con trỏ ?

Giá trị thực sự của con trỏ là địa chỉ ô nhớ mà sẽ chứa dữ liệu của biến động ứng với con trỏ, còn địa chỉ của bản thân con trỏ thì do chương trình dịch quyết định, bản thân người lập trình không cần quan tâm.

3.4.4.2. Những khác biệt trong cách sử dụng con trỏ trong các ngôn ngữ C, C++ và Pascal

- Trong khai báo:

Trong Pascal	Trong C và C++
Type <tên kiểu con trỏ> = ^<kiểu dữ liệu>; Var <danh sách biến con trỏ>:<tên kiểu con trỏ>; <danh sách biến con trỏ>:^<kiểu dữ liệu>; Trong đó <kiểu dữ liệu> có thể là integer, real, char, boolean, record... Ví dụ: Var intPtr : ^Integer;	<kiểu dữ liệu> * <tên biến con trỏ>; Trong đó <kiểu dữ liệu> có thể là int, char, void, double, long, struct, kiểu lớp, kiểu hàm... Ví dụ : int *a, *b ;

- Trong khởi tạo giá trị cho con trỏ:

Trong Pascal	Trong C và C++
Trong Pascal, khai báo biến con trỏ chỉ mới cung cấp thông tin về tên biến và kiểu dữ liệu con trỏ. Để khởi tạo con trỏ dùng thủ tục: New(tên biến con trỏ) cấp phát vùng nhớ cho nó. Khi không dùng con trỏ nữa thì sử dụng thủ tục: Dispose(<ten biến con trỏ>) để thu hồi lại bộ nhớ của nó.	- Con trỏ được khởi tạo và khởi gán đồng thời bằng lệnh gán <tên con trỏ> = <địa chỉ>; - Sử dụng thủ tục New và Delete để cấp phát và thu hồi bộ nhớ đối với con trỏ: <tên con trỏ> = New <tên kiểu dữ liệu>; Delete <tên con trỏ>; - Việc lấy địa chỉ của một biến để gán giá trị cho con trỏ có thể dùng toán tử một ngôi &, ví dụ: Int m, n = 10, *p; P = &n; M = *p;

- Trong sử dụng con trỏ:

Mục đích sử dụng con trỏ trong Pascal và C, C++ đều giống nhau, đều quan tâm đến ***biến động*** của con trỏ, chỉ khác ở cách viết. Trong Pascal,

biến động của con trỏ p được viết là p^, còn trong C, C++ được viết là *p. Ngoài ra trong Pascal không có toán tử một ngôi &, trong C và C++ lại dùng &<tên biến> để lấy địa chỉ của biến.

- Trong việc kết hợp con trỏ với các kiểu dữ liệu khác:

Con trỏ chỉ là tham chiếu đến địa chỉ vùng nhớ, còn dữ liệu tại vùng nhớ đó có kiểu gì thì phải được khai báo trong cú pháp khai báo con trỏ. Khái niệm kết hợp ở đây không phải là phép ghép, phép cộng mà là "thông qua", "dựa vào". Khi nói con trỏ p có kiểu integer thì được hiểu là nó trỏ đến vùng dữ liệu kiểu integer, con trỏ q có kiểu record(struct) thì được hiểu là nó trỏ đến vùng dữ liệu kiểu record(struct) với các trường xác định.

Ngoài ra trong C và C++ còn có một con trỏ đặc biệt là con trỏ void có cú pháp:

Void *<tên biến con trỏ>;

Khi khai báo bởi void, con trỏ sẽ được hiểu là con trỏ không kiểu và nó có thể nhận địa chỉ của đối tượng có kiểu bất kỳ.

3.5. DANH SÁCH LIÊN KẾT

3.5.1. Giới thiệu

Chúng ta đã xét các loại cấu trúc ngăn xếp, hàng đợi, đa thức, ma trận... là những loại danh sách tuần tự, có trật tự. Đa thức được sắp xếp theo trật tự số mũ, các ma trận được bố trí theo trật tự hàng cột. Chúng ta đã sử dụng mảng để cài đặt các danh sách tuần tự, trong phương pháp này chúng ta ngầm hiểu thứ tự của mảng cài đặt là thứ tự xử lý của các phần tử dữ liệu. Do cách cài đặt này mà mỗi khi chèn hay xoá một phần tử chúng ta phải dịch chuyển các phần tử khác để xác định lại trật tự của mảng. Điều này rất tốn thời gian trong trường hợp phải thường xuyên chèn - xoá.

Để khắc phục điều này, chúng ta sử dụng cấu trúc dữ liệu kiểu danh sách liên kết. Nếu như trong danh sách tuần tự, thứ tự các phần tử được ngầm hiểu, được ẩn, thì ngược lại trong danh sách liên kết, thứ tự các phần tử ở dạng hiện, được biểu diễn rõ.

Một danh sách liên kết là một dãy, một tập các nút, mỗi nút có 2 phần là phần *Dữ liệu* và phần *Liên kết*:

Dữ liệu	Liên kết

Ví dụ:

Giả sử ta có danh sách lưu trong mảng 1 chiều:

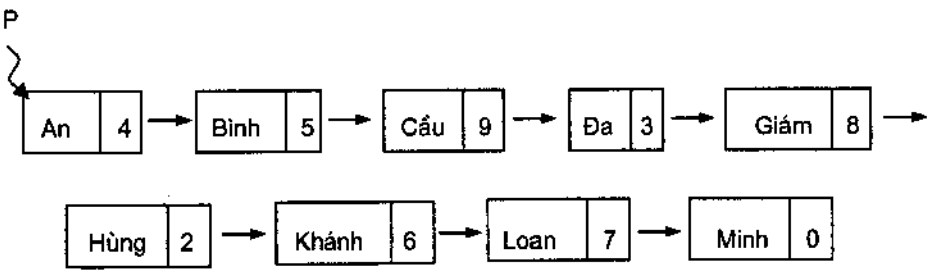
1	An
2	Khánh
3	Giám
4	Bình
5	Cầu
6	Loan
7	Minh
8	Hùng
9	Đa

Để thiết lập một trật tự theo vần ABC mà không cần phải dịch chuyển lại các phần tử, ta thêm 1 cột chỉ mối liên kết:

	Du_Lieu	Lien_ket
1	An	4
2	Khánh	6
3	Giám	8
4	Bình	5
5	Cầu	9
6	Loan	7
7	Minh	Nil
8	Hùng	2
9	Đa	3

Giá trị các phần tử trong cột *Liên kết* này sẽ trỏ đến các phần tử trong cột *Dữ liệu*. Nếu ta bắt đầu từ Du_Lieu[1] = AN, và ta đặt một biến con trỏ P có giá trị ban đầu là 1. Theo trật tự, sau An là Bình ở Du_Lieu[4], nên để trỏ đến Bình thì LienKet[1] = 4; sau Bình là Cầu ở Du_Lieu[5] nên LienKet[4] = 5,..., cứ thế cho đến sau Loan ở Du_Lieu[6] là Minh ở Du_Lieu[7] nên LienKet[6] = 7, sau Minh ở Du_Lieu[7] không có gì nên ta đặt LienKet[7] = Nil.

Ta có danh sách liên kết:



Với kiểu danh sách này khi chèn hay xoá một phần tử không cần phải dịch chuyển các phần tử khác trong danh sách.

Để cấu trúc được một danh sách liên kết, trước hết phải định vị được phần tử đầu tiên trong danh sách. Khi cho vị trí của một nút bất kỳ trong danh sách, ta có thể xác định được nút mà nó liên kết tới.

Một danh sách liên kết gồm:

- Một tập các nút

- Mỗi nút là một bản ghi trong đó có hai trường: trường dữ liệu và trường liên kết - con trỏ móc nối.

Phần dữ liệu của mỗi nút lưu trữ dữ liệu, phần con trỏ liên kết sẽ chỉ đến nút tiếp theo. Khi ở cuối danh sách, con trỏ liên kết không chỉ vào đâu cả, ta nói rằng nó là rỗng (Nil).

Nếu P là con trỏ, trỏ đến một nút nào đó trong danh sách liên kết, ta ký hiệu phần dữ liệu của nút này là Du_Lieu(p), phần liên kết là LienKet(p). Để tạo một danh sách rỗng ban đầu ta gán giá trị Nil vào đầu danh sách. Để kiểm tra một danh sách rỗng hay không ta chỉ việc xác định đầu danh sách có giá trị Nil hay không.

3.5.2. Cài đặt danh sách liên kết trên cơ sở mảng

Để giới thiệu các thao tác với danh sách liên kết, ta cần một số khai báo:

```
Const Max = 100;
Type Kieu_Ptu = <kiểu dữ liệu>;
    Kieu_LienKet = 0..Max; (* Kiểu liên kết *)
    NodeType = Record
                    Du_Lieu : Kieu_Ptu;
                    LienKet : Kieu_LienKet;
                    end;
    NodeArray = array[1.. Max] of NodeType;
Var Node : NodeArray;
    free :Kieu_LienKet; (*một biến kiểu liên kết chỉ đến nút tự do *)
    List: Kieu_LienKet ;(*Biến chỉ đến nút đầu tiên trong danh sách liên kết *)
```

3.5.2.1. Tạo một danh sách liên kết rỗng

```
Procedure Create(var List : Kieu_LienKet);
        Begin
                List : = 0 (*Nil*)
        End;
```

3.5.2.2. Khởi tạo vùng các nút tự do để lưu trữ các nút

```
Procedure InitializeStorage;
Var i : 1.. Max;
Begin
for i: = 1 to Max -1 Do
    Node[i]. LienKet : = i+1;  (*chỉ đến ô sau *)
    Node[Max ].LienKet: = 0 ;
    free : = 1
End;
```

Chỉ số mảng	Data	Next
List → 1	An	2
Free = 2 → 2		3
3		4
4		
5		5
6		6
7		7
8		8
9		0

Thủ tục này tạo một tập các nút rỗng để lưu trữ các phần tử của danh sách.

3.5.2.3. Thủ tục nhận một nút để tiếp nhận dữ liệu

```
Procedure GetNode (var P : Kieu_LienKet);
Begin       P: = free;
    If Free < > 0 then Free : = Node[free].LienKet
    Else Writeln(' vùng lưu trữ là rỗng' )
End;
```

Mỗi khi đưa thêm 1 phần tử mới vào danh sách phải thực hiện thủ tục này 1 lần và con trỏ free sẽ trỏ vào nút trống kế tiếp.

Ví dụ: Đưa An vào nút ở địa chỉ 1, biến List đầu danh sách chỉ vào 1, lúc đó vùng trống bắt đầu từ địa chỉ 2, free đặt bằng 2:

```
Node[1]. Du_Lieu : = 'An';
List : = free;
free : = Node[1].LienKet;
```

Chỉ số mảng	Data	Next
List → 1	An	2
free = 2 → 2		3
3		4
4		5
5		6
6		7
7		8
8		9
9		0

Sau đây là sơ đồ dữ liệu khi thêm các nút dữ liệu vào vùng lưu trữ.

Chỉ số mảng	Data	Next
List → 1	An	2
2	Bình	0
Free = 3 → 3		4
4		5
5		6
6		7
7		8
8		9
9		0

Thêm Node thứ 1

Chỉ số mảng	Data	Next
List → 1	An	2
2	Bình	3
3	Cầu	0
Free = 4 → 4		5
5		6
6		7
7		8
8		9
9		0

Thêm Node thứ 3

3.5.2.4. Giải phóng một nút, trả về vùng lưu trữ nút tự do

Khi ta xóa một nút, cần phải đưa nó về vùng lưu trữ các nút tự do để dùng tiếp sau này.

Ví dụ: Xóa tên "Bình" trong danh sách trên, ta có sơ đồ:

Chỉ số mảng	Data	Next
List → 1	An	3
Free = 2 → 2	Bình	4
3	Cầu	0
4		5
5		6
6		7
7		8
8		9
9		0

Tập nút trống bây giờ là:

Chỉ số mảng	Data	Next
Free = 2 → 2	Bình	4
4		5
5		6
6		7
7		8
8		9
9		0

Ta có thủ tục:

```
Procedure ReleaseNode( P: Kieu_LienKet );
    Begin Node[P]. LienKet : = free ;
        Free : = P
    End;
```

Ta không cần phải xoá dữ liệu hiện ở Node[p]. Du_Lieu. Sau này nếu nút được dùng lại, nó sẽ bị dữ liệu mới ghi đè.

3.5.2.5. Hàm kiểm tra danh sách có rỗng không

```
Function Empty(List : Kieu_LienKet): Boolean;
    Begin Empty : = (List = 0)
End;
```

3.5.2.6. Thủ tục xử lý dữ liệu các nút trong danh sách

Xử lý dữ liệu danh sách là ta truy nhập đến được từng nút và xử lý phần Du_Lieu. Có nhiều phép xử lý khác nhau, để đơn giản ta chọn công việc xử lý là hiện giá trị dữ liệu lên màn hình.

```
Procedure XuLy(List : Kieu_LienKet);
Var Pht: Kieu_LienKet; (* Biến liên kết chỉ đến node hiện thời đang xử lý *)
Begin
    Pht : = List;
While Pht < > 0 Do
```

```
        Begin
            Write(Node[Pht].Du_Lieu);
            Pht : = Node[Pht].LienKet
        End; Writeln
    End;
```

3.5.2.7. Thêm nút vào danh sách

a) Thêm vào đầu danh sách

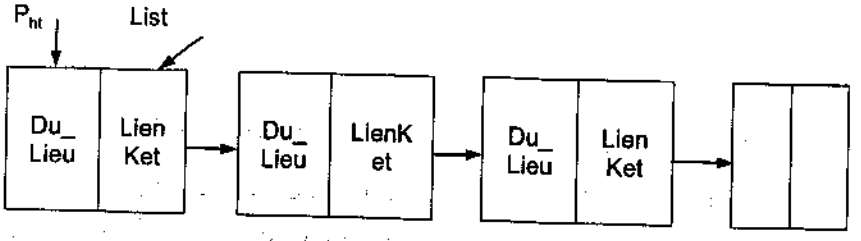
Để thêm một nút vào đầu danh sách, trước hết phải nhận một nút mới và lưu trữ dữ liệu vào đó. Ta cần thiết lập một thủ tục để nhận được một nút trong một vùng lưu trữ hiện có.

- Nhận nút, xác định P_{ht}
- Gán Du_Lieu [P_{ht}] : = <dữ liệu mới >;
- Chèn nút này vào đầu danh sách:

LienKet[P_{ht}]: = < Đầu danh sách>,

< Đầu danh sách > : = P_{ht}

Ta có sơ đồ:



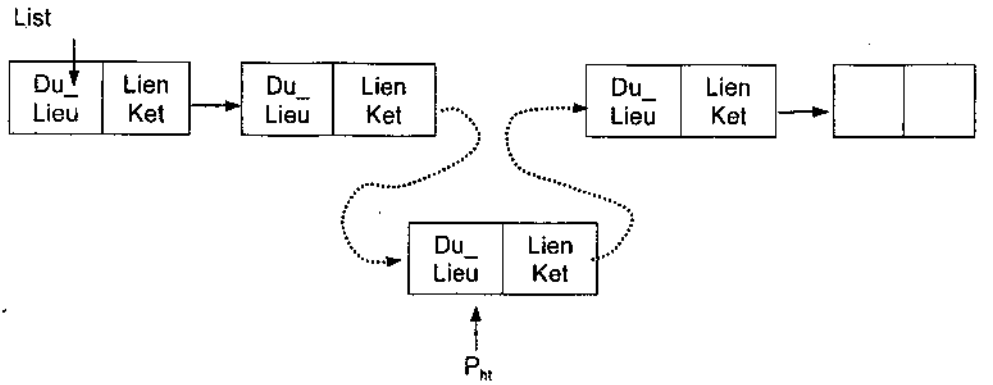
b) Chèn thêm một nút vào giữa danh sách sau một nút xác định

- Nhận một nút trống
- Đặt Du_Lieu[P_{ht}]: = <Dữ liệu>;

LienKet[Pht]: = LienKet[P_{truoc}];

LienKet [P_{truoc}]: = P_{ht} ;

Ta có sơ đồ:



Ta gộp hai trường hợp trên vào trong một thủ tục như sau:

```
Procedure Insert (var List: Kieu_LienKet ; Item: Kieu_Ptu;
                                  P_truoc: Kieu_LienKet);
(*P_truoc: biến liên kết chỉ đến vị trí trước vị trí cần chèn vào *)
Var
   Pht: Kieu_LienKet(* biến liên kết chỉ đến nút mới cần được chèn vào*)
Begin Getnode(P_ht);
Node[P_ht]. Du_Lieu : = Item;
   If P_truoc = 0 Then
     Begin Node [P_ht].LienKet : = List ;
           List : = P_ht
     End
       Else
         Begin
           Node[P_ht].LienKet : = Node[P_truoc]. LienKet;
           Node [P_truoc].LienKet: = P_ht
         End
End;
```

Chèn thêm vào đầu danh

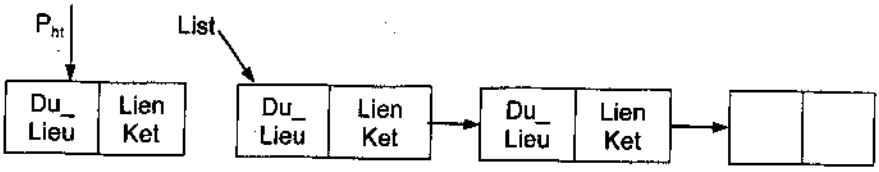
Chèn vào giữa danh sách, sau Node [P_{truoc}]

3.5.2.8. Xoá nút

a) Xoá ở đầu danh sách

- Đặt P_{ht} : = List
- Đặt List chỉ vào nút thứ hai: List : = Node[P_{ht}].LienKet

- Đưa nút đầu về tệp các nút rỗng:

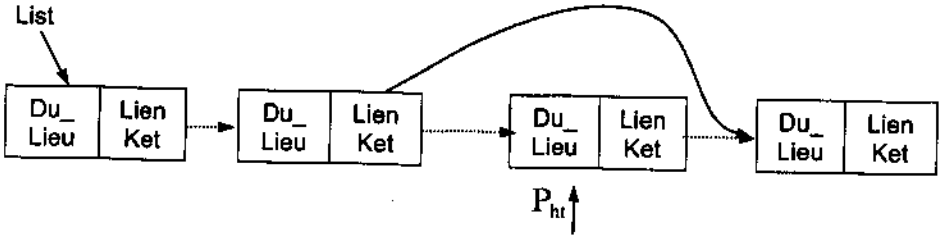


b) Xoá nút hiện thời

Chỉ cần đặt lại: $P_{ht} := LienKet[P_{truoc}]$

$LienKet[P_{truoc}] := LienKet[P_{ht}]$

Ta có sơ đồ:




```
Procedure Delete(var List: Kieu_LienKet; Ptruoc : Kieu_LienKet);
   Var Pht: Kieu_LienKet;
Begin
   If Empty(List) then halt
      Else
        Begin
           If Ptruoc = 0 then (*Xóa đầu danh sách *)
                 Begin Pht : = List;
                       List : = Node[Pht].LienKet
                 End
             Else
               Begin (*Xóa giữa danh sách *)
                       Pht : = Node[Ptruoc].LienKet;
                       Node[Ptruoc].LienKet : = Node[Pht].LienKet
               End;
            ReleaseNode(Pht) ;(*Giai phong nut *)
        End;
End;
```

3.5.3. Cài đặt danh sách liên kết trên cơ sở con trỏ

Chúng ta dùng các bản ghi và mảng để lưu trữ các nút, mỗi bản ghi có hai phần: Dữ liệu - Du_Lieu và Liên kết - LienKet, các nút liên kết theo chỉ số mảng. Khác với cách cài đặt trên, cách cài đặt trên cơ sở con trỏ là liên kết con trỏ chứ không phải là liên kết theo chỉ số mảng.

Trong mục giới thiệu về con trỏ và dữ liệu động, ta đã biết mỗi biến con trỏ được khai báo bằng:

```
TYPE <Tên kiểu con trỏ> = ^<Kiểu dữ liệu>;
VAR <Biến con trỏ> : <Tên kiểu con trỏ>;
```

Đối với danh sách liên kết, ta có thể khai báo:

```
Type   Kieu_Ptu = <kiểu dữ liệu>;
       Kieu_ConTro = ^ NodeType;
       NodeType = Record
                    Du_Lieu: Kieu_Ptu;
                    LienKet: Kieu_ConTro
                    End;
Var
    List : Kieu_ConTro;
```

***Chú ý**: Khi dùng con trỏ, trong Pascal đã có 2 thủ tục New và Dispose để tạo con trỏ mới, giải phóng con trỏ, vì vậy không phải viết các thủ tục GetNode và ReleaseNode như khi cài đặt trên cơ sơ mảng.*

Các thủ tục đối với danh sách liên kết trên cơ sở dùng con trỏ, về mặt thuật toán vẫn giống như trong trường hợp dùng mảng, chỉ khác ở phần khai báo con trỏ.

3.5.3.1. Thiết lập danh sách liên kết rỗng

```
Procedure Create(var List: Kieu_ConTro);
Begin List : = Nil End;
```

3.5.3.2. Kiểm tra danh sách rỗng

```
Function Empty(List: Kieu_ConTro ):boolean;
Begin Empty : = ( List = Nil) End;
```

3.5.3.3. Xử lý danh sách

```
Procedure Xuly(List: Kieu_ConTro );
Var
    Pht: Kieu_ConTro ; (* con trỏ trỏ đến nút hiện thời được xử lý *)
Begin
   Pht: = List;
While Pht < > Nil Do
       Begin
           Writeln(Pht^.Du_Lieu);
           Pht: = Pht^.LienKet
       End
End;
```

3.5.3.4. Xoá nút

```
Procedure Delete(var List: Kieu_ConTro ;Ptruoc : Kieu_ConTro );
Var Pht: Kieu_ConTro ;
Begin
        If Empty(List) Then Halt
        Else
        Begin If Ptruoc = Nil then (*xoá nút đầu tiên*)
                    Begin
                        Pht : = List ;
                        List : = Pht^.LienKet
                    End
                    Else (* xoá nút hiện thời*)
                      Begin
                        Pht : = Ptruoc ^.LienKet;
                        Ptruoc^.LienKet : = Pht^.LienKet
                      End;
                    Dispose (Pht)
            End;
   End;
```

3.5.3.5. Chèn thêm nút

```
Procedure Insert( var List: Kieu_ConTro; Item: Kieu_Ptu;
                                  P_truoc: Kieu_ConTro );
(*P_truoc : Con trỏ trỏ đến vị trí trước vị trí cần chèn vào *)
Var
    P_ht: Kieu_ConTro (* Con trỏ trỏ đến nút mới cần được chèn vào*)
Begin New(P_ht);
    P_ht^. Du_Lieu : = Item;
    If P_truoc = Nil Then
          Begin P_ht^.LienKet : = List ;
                List : = P_ht
          End
          Else
             Begin
                  P_ht^.LienKet : = P_truoc^.LienKet;
                  P_truoc.LienKet: = P_ht
             End
End;
```

Chèn thêm vào đầu danh sách

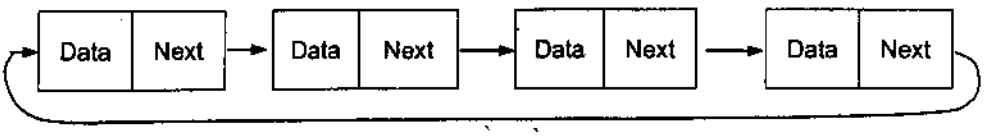
Chèn vào giữa danh sách, sau P_{pred}

3.5.4. Các loại danh sách liên kết khác

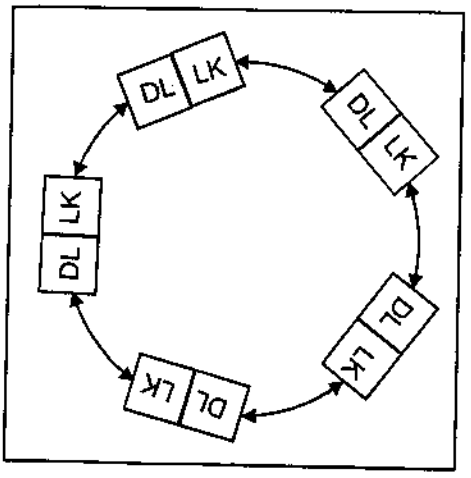
3.5.4.1. Danh sách liên kết vòng tròn

a) Khái niệm

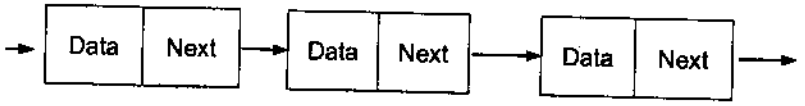
Danh sách liên kết kiểu vòng tròn là kiểu danh sách có phần tử cuối cùng lại liên kết với phần tử đầu tiên của danh sách:



Nếu ta vẽ lại các nút của danh sách theo kiểu vòng tròn thì không còn phân biệt đâu là nút đầu, đâu là nút cuối, nghĩa là các nút đều bình đẳng như nhau, mỗi nút đều có thể là đầu, có thể là cuối:



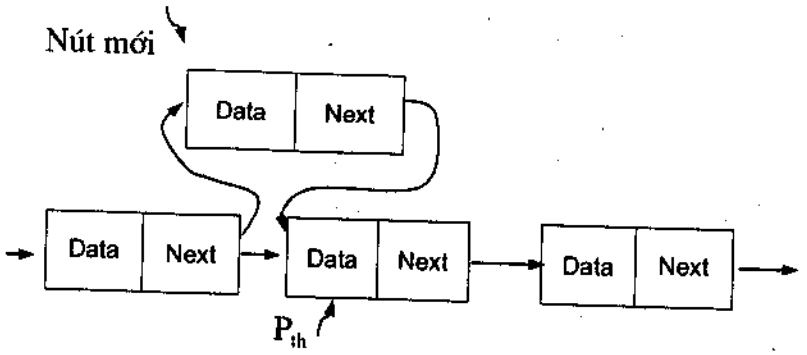
Vì các phần tử nút của danh sách liên kết vòng là bình đẳng nên để khảo sát các thủ tục đối với danh sách này chỉ cần xét trên mô hình của 3 nút liền kề:



b) Chèn thêm nút vào danh sách liên kết vòng

Tại một bộ 3 nút bất kỳ, ta sử dụng một con trỏ P_{ht} trỏ tới nút hiện thời ở giữa.

Giả sử cần chèn thêm một nút mới vào trước nút hiện thời.



Khi đó:

$P^{\wedge}_{truoc}$.Next : = <địa chỉ của nút mới>;

<Nút mới>.Next: = $P^{\wedge}_{truoc}$.Next

Trường hợp chèn thêm một nút mới vào sau nút hiện thời thì:

$P^{\wedge}_{th}$, Next : = <địa chỉ của nút mới>;

<Nút mới>.Next: = $P^{\wedge}_{ht}$.Next

(Sinh viên có thể tự viết được hai thủ tục chèn thêm phần tử cho 2 trường hợp trên).

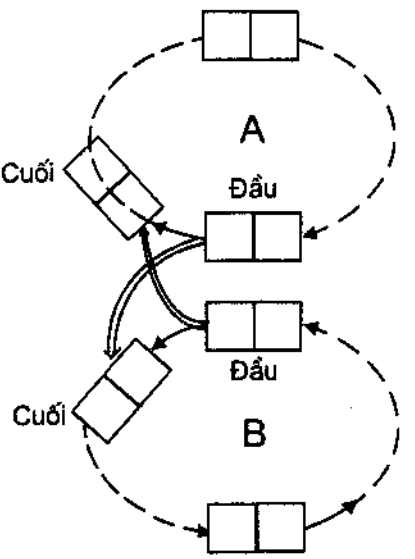
c) Nối hai danh sách liên kết vòng

Giả sử có 2 danh sách liên kết vòng A và B, cần viết câu lệnh để nối 2 danh sách đó thành một danh sách. Ta có thể ngắt bất kỳ ở một điểm nào của từng danh sách để nối, để xác định ta chọn một điểm ngắt như trên hình vẽ và coi một nút là đầu danh sách A, một nút là cuối danh sách A, một nút là đầu danh sách B, một nút là cuối danh sách B. Như trên hình vẽ, đầu A sẽ nối sang cuối B, đầu B sẽ nối sang cuối A, ta có các câu lệnh hoán đổi địa chỉ <Đầu A.Next> < = > <Đầu B. Next> như sau:

<Trung gian> : = Đầu A.Next;

Đầu A.Next : = Đầu B.Next;

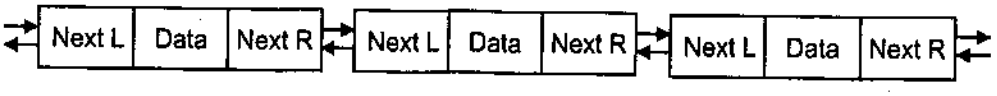
Đầu B.Next: = <Trung gian>;



3.5.4.2. Danh sách liên kết đôi

Danh sách liên kết đôi là kiểu danh sách mà một nút có 3 thành phần : Liên kết bên trái (ký hiệu NextL), Dữ liệu (Data), Liên kết bên phải (ký hiệu NextR).

Khi để nhiều nút ta sẽ có sơ đồ:



Xét thủ tục xóa một nút, giả sử xóa nút hiện thời(nút ở giữa), để tiện viết câu lệnh ta sử dụng con trỏ P_{ht} chỉ nút cần xóa, P_{sau} chỉ nút ở sau nút cần xóa, P_{truoc} chỉ nút ở trước nút cần xóa.

$P^{\wedge}_{truoc}$. NextR : = $P^{\wedge}_{ht}$. NextR;

$P^{\wedge}_{sau}$. NextL : = $P^{\wedge}_{ht}$. NextL

Dispose(P_{ht});

3.5.5. Ứng dụng của danh sách liên kết

a) Chương trình ví dụ

Ta có thể ghép nối các function và procedures ở mục 3.5.2 trên đây, bổ sung thêm một số biến và dữ liệu cần thiết để làm thành một chương trình ví dụ về vận dụng danh sách liên kết quản lý một danh sách sinh viên.

```
Program dslket;
uses dos,crt;
const max=100;
type kieuptu=string;
    kieulk=0..max;
    nodetype=record
            dl:kieuptu;
            lk:kieulk
           end;
```

```
    nodeArray=array[1..max] of nodetype;
var node:nodeArray;
   Ptr,list,free:kieulk;
   Item:string;
   m,n,chon,chucnang:byte;
   traloi:char;
   bg:nodeType;
   f:file of Nodetype;
Procedure create(var list:kieulk);
   begin  list:=0 end;
Procedure InitializeStorage;
 var i:1..max;
 begin
  for i:=1 to max-1 do node[i].lk:=i+1;
  node[max].lk:=0;
  free:=1
 end;
Procedure Getnode(var p:kieulk);
 begin
   p:=free;
   if free<>0 then free:=node[free].lk
           else writeln('Vung luu tru Rong')
 end;
Function Empty(list : kieulk):boolean;
  begin  empty:=(list = 0) end;
Procedure insert(var list:kieulk;Item:kieuptu;Ptr:kieulk);
 var Pht:kieulk;
 Begin
     Getnode(Pht);
     node[Pht].dl := item;
     if Ptr=0 then begin node[Pht].lk :=list;
```

```
                list:=Pht
            end
        else
            begin node[Pht].lk:=node[Ptr].lk;
                node[Ptr].lk:=Pht
            end
    end;
    procedure Change;{ sửa đổi thông tin}
    begin
      {sinh viên tự thêm các mã lệnh để sửa đổi thông tin tùy ý}
    end;
Procedure HienDulieu;
    var Pht:kieulk;
    Begin
        Pht:=list;
        while Pht <> 0 do
          begin writeln(node[Pht].dl:25);
              Pht:=node[Pht].lk
          end;
       readln
    end;
Procedure  ReleaseNode(p:kieulk);
    begin
        node[p].lk := free;
        free:=p
    end;
Procedure Delete(var list:kieulk;Ptr:kieulk);
    var Pht:kieulk;
    Begin
     If empty(list) then halt
        else begin
```

```
            if Ptr=0 then
               begin Pht:=list;
                  list:=node[Pht].lk
               end
              else
        begin Pht:=node[Ptr].lk;
         node[Ptr].lk:=node[Pht].lk
        end;
        ReleaseNode(Pht);
           end
  End;
Begin (*Chuong trinh chinh*)
 write('Nhap so phan tu cua danh sach:');
   readln(n);
   chon:=6;
    traloi:='Y';
  while upcase(traloi)='Y' do
   begin
    gotoxy(1,1);
writeln('=================================');
writeln('0.InitializeStorage                 ');
writeln('1.Khoi tao danh sach                ');
writeln('2.Hien du lieu                      ');
writeln('3.Them vao danh sach                ');
writeln('4.Sua thong tin d.sh                ');
writeln('5.Xoa                               ');
writeln('6.Ket thuc                          ');
writeln('bam so 0,1,2,3,4,5,6 để chọn        ');
writeln('=================================');
```

```
        readln(chon);
        clrscr;
         case chon of
          0:InitializeStorage;
          1:create(list);
          2:HienDulieu;
          3:insert(list,Item,Ptr);
          4:Change;
          5:Delete(list,Ptr);
          6:halt;
         end;
    writeln('Co lam tiep khong?(Y/N)                    ');
    writeln('==================================');
       clrscr;
       readln(traloi);
     end;
      readln
     end.
```

b) Sử dụng danh sách liên kết để viết chương trình lọc ra các xâu ký tự bắt đầu bằng một chữ cái từ A đến Z.

Chương trình sẽ đọc các chữ cái lưu trong file text đã có từ trước trên đĩa từ, xác định xâu bắt đầu từ chữ cái đó cho đến khi gặp dấu cách.

```
Program textCorcordance(input,textfile,ouput);
const  MaxString =15;
Type  Kieu_Ptu = String;
      Kieu_ConTro =^NodeType;
   NodeType = Record
                  Du_Lieu :Kieu_Ptu;
                  LienKet :Kieu_ConTro;
                end;
```

```
    ArrayOfPointer = Array['A'..'Z'] of Kieu_ConTro;
 Var    P_truoc :Kieu_ConTro;
        TextFile : Text;
        Word :String;
        List: ArrayOfPointer;
       Ch:char;
       Found,MoreWords: Boolean;
       i:integer;
Procedure CreateList(Var List :Kieu_ConTro);
   Begin List:= Nil End;
Function EmptyList(List : Kieu_ConTro): Boolean;
  Begin EmptyList := (List = Nil) end;
Procedure Xuly(List:Kieu_ConTro);
 Var P_ht :Kieu_ConTro;
 Begin P_ht := List;
   While  P_ht <> Nil Do
    Begin Write(P_ht^.Du_Lieu);
        P_ht := P_ht^.LienKet
    End;
    Writeln
 End;
Procedure Search(List:Kieu_ConTro;Item:Kieu_Ptu;
                        var  P_truoc:Kieu_ConTro;var found:boolean);
   Var  P_ht:Kieu_ConTro;
        Done:boolean;
   Begin
      P_ht:= List; P_truoc := Nil;found := false;Done:= false;
    While not Done and (P_ht <> Nil) Do
            If  P_ht^.Du_Lieu >= Item Then
```

```
                    Begin     Done := True;
                              Found :=(P_ht^.Du_Lieu = Item)
                    End
                                   Else
                    Begin
                        P_truoc := P_ht;
                        P_ht := P_ht^.LienKet
                End
        End;
Procedure Insert(Var List:Kieu_ConTro;Item:Kieu_Ptu;
            P_truoc:Kieu_ConTro);
    Var  P_ht :Kieu_ConTro;
    Begin
        new(P_ht);
        P_ht^.Du_Lieu := Item;
        If P_truoc = Nil Then
          Begin P_ht^.LienKet := List;
              List := P_ht
          End
                    Else
         Begin
           P_ht^.LienKet := P_truoc^.LienKet;
         P_truoc^.LienKet := P_ht
        end
    end;
Procedure GetAWord(Var TextFile:Text;Var Word:String;
            Var Morewords :Boolean);
  Var
   Ch :char;
   i :integer;
```

```
Begin
 MoreWords := false;
 While not MoreWords and not eof(TextFile) Do
  Begin
     read(TextFile,Ch);
     MoreWords := (Ch IN ['A'..'Z']);
  End;
  IF MoreWords Then
   Begin
    Word[1] := Ch;
    For i:=2 to MaxString Do word[i] := ' ';
    i:=1;
    If not eof(TextFile) Then read(TextFile,Ch);
    While (Ch IN ['A'..'Z'] ) and not eof(TextFile) Do
       Begin
        i:=i+1;
        word[0] := Ch;
        word[i]:=Ch;
        read(TextFile,Ch)
       end
   end
 end;
Begin (* Chương trình chính *)
  For Ch := 'A' to 'Z' Do CreateList(List[Ch]);
  Assign(TextFile,'p235.txt');
  Reset(TextFile);
  GetAWord(TextFile,Word,MoreWords);
  WRITELN('Ch =',Ch);
```

```
    readln;
    (* Lập xâu các chữ cái *)
  While MoreWords Do
   Begin
      Search(List[word[1]],Word,P_truoc,found);
      IF not found Then Insert(List[word[1]],word,P_truoc);
      GetAWord(TextFile,Word,MoreWords);
   end;
 For Ch := 'A' to 'Z' Do
  If not EmptyList(List[ch]) Then
    Begin
      Writeln('Những từ bắt đầu bằng ',Ch,':');
      Xuly(List[Ch]);
      writeln;
    end;
  readln
end.
```

3.6. NGĂN XẾP VÀ HÀNG ĐỢI LIÊN KẾT

3.6.1. Ngăn xếp liên kết

Khi chỉ có nút đầu tiên trong danh sách liên kết được truy nhập trực tiếp thì ta có một cơ chế làm việc như là ngăn xếp. Trên cơ sở con trỏ, ta có thể khai báo cho một ngăn xếp liên kết như sau:

```
Type Kieu_Ptu = <kiểu dữ liệu>;
        Kieu_ConTro = ^ StackNode;
        StackNode = Record
                Du_Lieu : Kieu_Ptu;
                LienKet: Kieu_ConTro
                End;
```

```
        StackType = Kieu_ConTro;
    Var
        Stack: StackType;
```

Các thao tác chèn, lấy ra đều được thực hiện ở đỉnh ngăn xếp.

```
Procedure  Pop(var stack: StackType; var  Item: Kieu_Ptu);
(*Thủ tục lấy ra Item từ đỉnh ngăn xếp liên kết *)
Var  Pht: Kieu_ConTro; (* chỉ nút ở đỉnh Stack*)
Begin if   Empty(stack) then  halt (*rong*)
        Else
        Begin
           Item: stack ^.Du_Lieu;
           Pht:=Stack;
           Stack:= stack^.LienKet;
              Dispose (Pht)
        End
   End;
Procedure  Push(Var  Stack: StackType ; Item: Kieu_Ptu);
   Var
      Pht: Kieu_ConTro; (* chỉ nút ở đỉnh*)
   Begin
        New (Pht);
        Pht^.Du_Lieu:=Item;
        Pht^. LienKet:= stack;
        Stack := Pht
   End;
```

Ví dụ: Chương trình dùng ngăn xếp để chuyển đổi biểu diễn thập phân của một số nguyên dương sang biểu diễn cơ số hai.

```
Program  Coso10_2;
  Type  Kieu_Ptu= integer;
```

```
        Kieu_ConTro=^ StackNode;
        StackNode= Record
          Du_Lieu: Kieu_Ptu ;
          LienKet: Kieu_ConTro ;
          End ;
        StackType= Kieu_ConTro ;
Var
      So,sodu: integer
      Stack: StackType:
      Traloi: char;
Procedure  Create(var stack: StackType);
      Begin  stack: = Nil       End;

Function  Empty(stack: StackType): Boolean;
      Begin  Empty := (stack =Nil)        End;

Procedure  Pop(var Stack: StackType; Var Item: Kieu_Ptu);
Var
    Pht: Kieu_ConTro;
Begin
   if  Empty(stack)  then  Halt (* layra tu ngăn xep rong*)
      Else
          Begin
                Item: =Stack^.Du_Lieu;
                Pht: =stack;
                Stack:= stack^ LienKet;
                Dispose(pht)
           End
End;
Procedure  Push(var Stack : StackType; Item: Kieu_Ptu);
```

```
    Var  P_ht: Kieu_ConTro;
    Begin    New(P_ht);
             P_ht^.Du_Lieu:= Item;
             P_ht^.LienKet:= stack;
             Stack:= P_ht
    End;
Begin (*Chương trình chính*)
    Repeat
        Write ('Dưa vào số để chuyển đổi:');
        Readln(so);  Create(stack);
        While  so < > 0
        Begin  sodu :=  sodu Mod 2:
                Push(stacks,sodu);
                So: = so div  2
        End (*while*) ; writeln('biểu diễn cơ số 2: ');
        While  not  Empty(stack)  Do
        Begin
              Pop(stack,sodu);
              Write (sodu:1)
        End;
            Writeln;
            Write ('Làm tiếp  hay không(Y/N)?); Readln(traloi) ;
            Until  not (traloi in ['Y','y'])
End.
```

3.6.2. Hàng đợi liên kết

Trong hàng đợi liên kết, phần tử đầu tiên của danh sách sẽ là đầu hàng đợi, cuối danh sách là cuối hàng đợi. Ta sử dụng hai con trỏ cho đầu và cuối hàng đợi:

```
Type      Kieu_Ptu= <kiểu dữ liệu>;
          Kieu_ConTro=^ QueueNode;
          QueueNode= Record
                        Du_Lieu: Kieu_Ptu
                        LienKet: Kieu_ConTro
                     End;
          QueueType  = record
                        Front,
                        Rear : Kieu_ConTro
                     End;
     Var  Q: QueueType
```

Các thủ tục của hàng đợi liên kết cũng giống như ngăn xếp đã xét ở trên.

3.7. PHÉP ĐỆ QUY

3.7.1. Khái niệm

Đệ quy là hiện tượng một thủ tục hay một hàm tham trỏ đến chính nó.

Một thủ tục đệ quy hay hàm đệ quy gồm hai phần:

1. Phần neo (anchor) trong đó tác động của hàm hay thủ tục được đặc tả cho một hay nhiều tham số.

2. Phần đệ quy, trong đó tác động cần được thực hiện cho giá trị hiện thời của các tham số được định nghĩa bằng các tác động hay giá trị được định nghĩa trước đó.

Ví dụ: Để tính giai thừa có thể mô tả đệ quy.

- Ở đây phần ***neo*** là 0! =1

đệ quy là với n>0 thì n! =n(n-1)!

N!= n*(n-1)!

(n-1)! = (n-1)*(n-2)!

. . . . . .

1!=1x0!

0!=1 cuối cùng đi đến phần neo.

- Viết hàm đệ quy sau:

```
Function  Giaithua(n: integer ) :integer;
  Begin  if  n=0  then  Giaithua:=1
      Else
        Giaithua :=  n*giaithua (n-1)
   End;
```

Ví dụ 2: Số Fibonaci

1

1

2 với n> 2 mỗi số sau là tổng của hai số trước nó

3

5

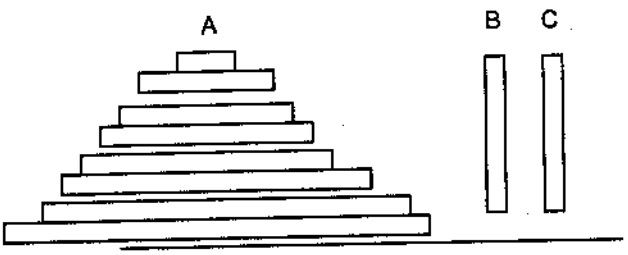
8 Neo { Fib(1)=1 ; Fib(2)=1 }

Với n>2, Fib(n)=Fib(n-1)+Fib(n-2)

```
Function Fib(n: integer): integer;
  Begin
     If  (n<=2)  then   Fib :=1
                 Else
                   Fib:= Fib(n-1)+ Fib(n-2)
     End;
```

3.7.2. Ví dụ về phép đệ quy

Tháp Hà Nội:



Cần phải chuyển các đĩa kích thước khác nhau từ cọc bên trái sang cọc bên phải, với các luật sau:

1. Khi di chuyển một đĩa, nó phải được đặt vào một trong ba cọc đã cho
2. Mỗi lần chỉ di chuyển một đĩa và phải là đĩa ở trên cùng
3. Đĩa lớn hơn không được phép nằm trên đĩa nhỏ hơn. Cho n là số đĩa.

Ta làm đệ quy như sau:

Nếu n = 1, việc chuyển đĩa từ A → C là giải được.

1. Giả sử đã chuyển được đối với n-1 đĩa trên cùng ở A → B
2. Chuyển đĩa thứ n → cột C (C là cọc phụ)
3. Chuyển (n-1) đĩa từ B → C, dùng A là cọc phụ:

Chương trình:

```
Program Thap_Hanoi;
 Const coc1 ='A';
      coc2 ='B';
      coc3 ='C';
 Var Sodia : integer;
 Procedure Move(n:integer;Dau,Phu,Cuoi :char);
  Begin
  If n=1 then  writeln('Chuyen dia tu coc ',Dau,' sang coc',Cuoi)
     else
      begin
       Move(n-1,Dau,Cuoi,Phu);
       Move(1,Dau,' ',cuoi);
       Move(n-1,Phu,Dau,Cuoi);
      end
  end;
Begin (*Chuong trinh chinh *)
 write('Cho so dia : ');readln(sodia);
 Move(sodia,coc1,coc2,coc3);
 readln
end.
```

Nhận xét:

- Đệ quy tiện lợi về lập trình nhưng trong nhiều trường hợp không lợi về thời gian tính:

Khi tính giai thừa theo đệ quy, độ phức tạp là T(n) = O(n); tính theo không đệ quy cũng T(n) = O(n). Ta xét lại hàm đệ quy tính giai thừa:

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

- Tính độ phức tạp:

+ Với n = 0, hàm phải thực hiện 2 phép toán (so sánh n = 0 và gán GT: = 1);

Tức là T(0) = 2

+ Với n > 0 thời gian tính T(n) bằng 2 + thời gian tính n*GT(n-1)

T(n) = 2+T(n-1)

Tương tự T(n-1) = 2+T(n-2)

.....

T(1) = 2+T(0)

Hay T(n) = 2+2+...+2+T(0) = 2+2+...+2 (n + 1 lần) = 2*(n+1)

Hay T(n) = O(n)

Nhưng trong thủ tục đệ quy tính số Fibonaci:

$T(n) \geq 2^{n/2}$ với $\forall n>2$.

Thật vậy

Với n>2

T(n) = 2+T(n-1)+T(n-2)

Với n>3

T(n-1) = 2+T(n-2)+T(n-3)

Thay vào T(n): T(n) = 4 +2T(n-2)+T(n-3)≥ 2T(n-2)

$$T(n) \geq 2T(n-2)\ \forall n > 3$$

Truy hồi, với n >5 : $T(n-2) \geq 2T(n-4)$

$T(n) \geq 2T(n-2) \geq 2*(2T(n-4)) = 4T(n-4) \geq 2^{n-2/2}T(2)$ nếu n chẵn;

$T(n) \geq 2T(n-2) \geq 2*(2T(n-4)) = 4T(n-4) \geq 2^{(n-1)/2}T(1)$ nếu n lẻ.

Vì T(1) = T(2) = 2 nên trong cả hai trường hợp ta đều có

$$T(n) \geq 2^{n/2}\ \forall n > 2$$ - độ phức tạp luỹ tiến.

Bảng dưới đây là kết quả thống kê thời gian chạy thử hàm fib(n) tính theo mili giây.

N	10	15	20	22	24	26	28	30
Thời gian	5	69	784	2054	2464	14121	36921	96494

3.8. CÂY - TREE

3.8.1. Khái niệm

Cấu trúc dữ liệu kiểu Tree là tập hợp các nút (hay gọi là đỉnh) và những cạnh có hướng nối với các cặp nút với nhau.

- Nút gốc ① không có cạnh nào đến nó, chỉ có từ nó đi ra
- Nút con - nút đi ra từ nút gốc
- Lá không có cạnh nào đi ra.

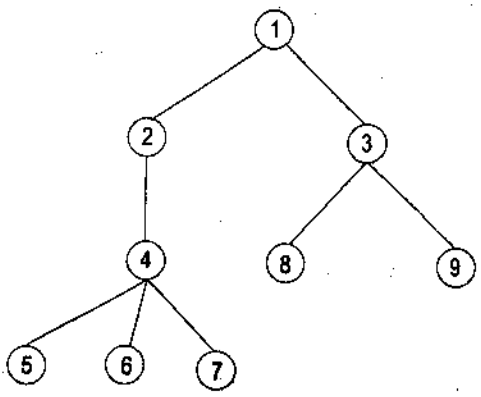
Ví dụ 1: Cho cây như hình bên.

Nút 1 là gốc

nút 2, 3 là nút con và 1 là nút bố

nút 2, 3 còn gọi là nút anh em

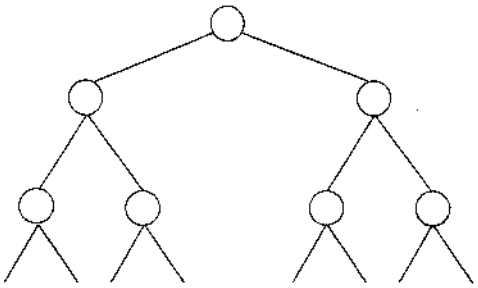
nút 7, 8, 9 gọi là lá



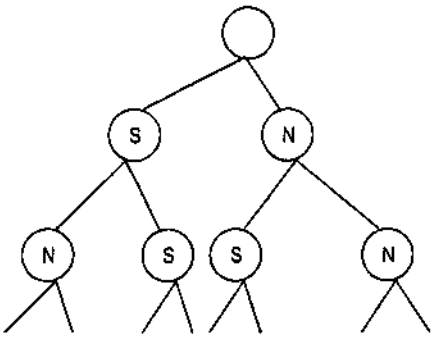
Ví dụ 2: Cây gia hệ.

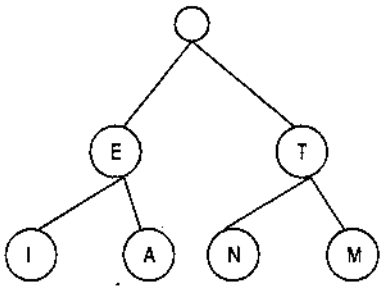
3.8.2. Cây nhị phân

Cây nhị phân là cây trong đó mỗi nút có nhiều nhất hai nút con đi ra.



Cây nhị phân là công cụ mô phỏng tốt các quá trình có 2 khả năng xảy ra: tung hứng đồng tiền (sấp ngửa) và morse:

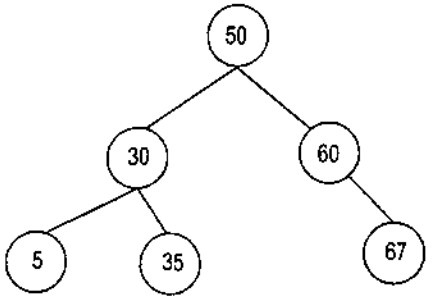




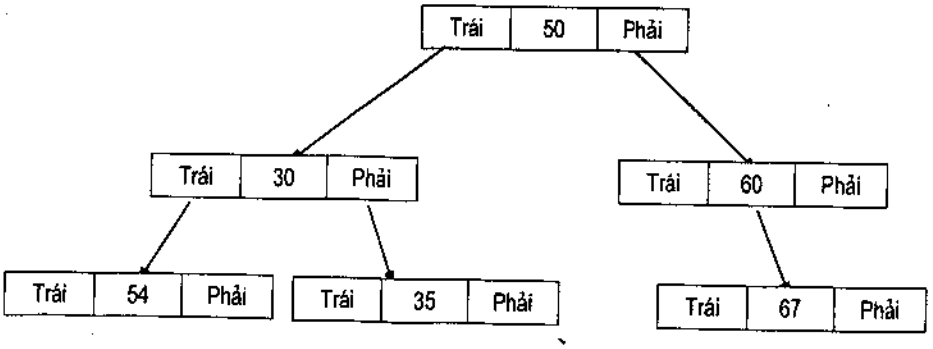
Có thể truy nhập đến một nút bất kỳ của cây nhị phân nếu ta bảo trì một con trỏ chỉ đến nút gốc của cây. Dùng các bản ghi để biểu diễn các nút và các con trỏ. Ta có khai báo cho cây nhị phân:

```
Type
    Kieu_Ptu= <kiểu dữ liệu>;
    Treepointer= ↑ TreeNode;
    TreeNode= Record
                  Du_Lieu: Kieu_Ptu;
                  Lchild,Rchild,TreePointer
              End;
Var  Goc: TreePointer;
```

Ví dụ: Cây nhị phân



Có thể được biểu diễn:



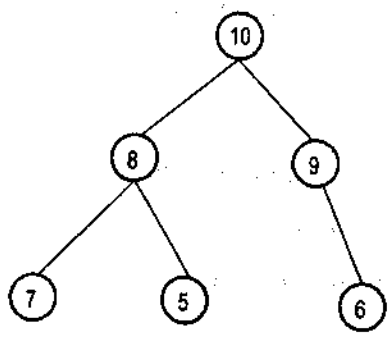
3.8.3. Khối (Heap)

Khối là một loại cây nhị phân đặc biệt có các đặc điểm:

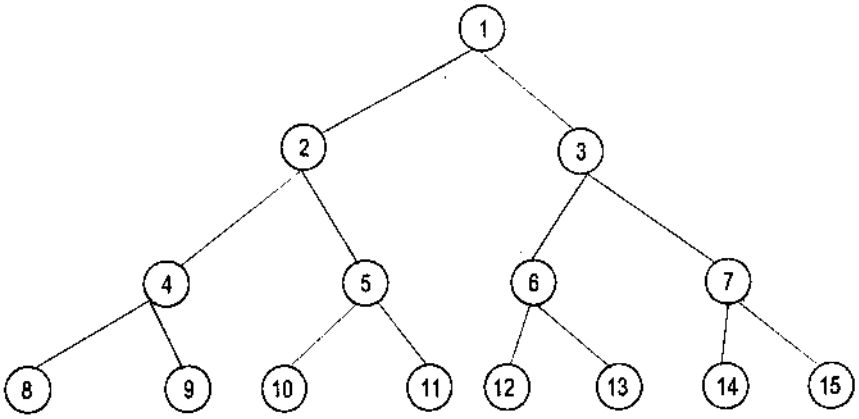
- Nó là một cây đầy đủ, nghĩa là các lá trên cây nằm nhiều nhất ở hai mức liên tiếp nhau và các lá ở mức dưới nằm ở những vị trí bên trái nhất (leftmost positions).

- Giá trị trong mỗi nút lớn lớn hơn các giá trị ở các mục con.

Ví dụ: Khối như hình sau:



Để cài đặt khối, ta có thể dùng cấu trúc liên kết như đối với cây nhị phân, song có thể dùng mảng, ta đánh số thứ tự từ trên xuống, ở mỗi mức các nút được đánh số từ trái qua phải:



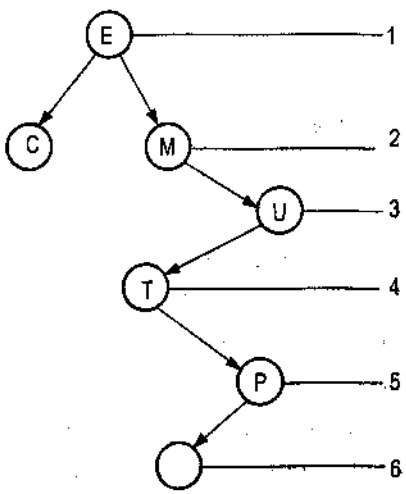
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T(i)															

Tính đầy đủ của khối bảo đảm cho các mục dữ liệu được lưu trữ trong các vị trí liên tiếp ở đầu mảng. Có thể khai báo mảng Heap như sau:

```
const Sonut =... {số nút cực đại trong khối}
Type
    kieuphantu =... { kieu dữ lieu}
    Heap: Heaptype;
```

Trong cách cài đặt mảng như vậy việc tìm con của nút thứ i sẽ ở vị trí 2*i và 2*i+1, bố của nút thứ i nằm ở vị trí (i div 2).

Đối với các loại cây nhị phân khác, cài đặt mảng là không hiệu quả:



3.8.4. Số lượng nút của một cây nhị phân

Ta có thể xác định số lượng tối đa các nút của một cây nhị phân:

3.8.4.1. Bổ đề 1

1. Số lượng tối đa các nút ở mức i của cây nhị phân là 2^{i-1} với $i \geq 1$ và

2. Số lượng tối đa các nút trong cây nhị phân có K mức là 2^k-1, $k \leq 1$.

a) Chứng minh quy nạp

- Gốc ở mức i = 1 như vậy $2^{i-1} = 2^0 = 1$ là đúng !

- Giả sử kết quả trên đúng với $\forall j$, $1 \leq j \leq i$, số lượng các nút ở mức j là 2^{j-1}.

Ta cần chứng minh điều này đúng cho i = j+1.

Ở mức i-1, theo giả thiết là có 2^{i-2} nút, như vậy, cứ mỗi nút ở mức này lại có tối đa là 2 nút ở mức i tức là số nút ở mức i tối đa bằng 2 lần số nút ở i-1:

$$2^{i-2} \times 2 = 2^{i-1}$$

b) Để tính số nút tối đa trong cây K mức ta lấy:

$$\sum_{i=1}^{k} 2^{i-1} = 2^0 + 2^1 + 2^2 + 2^3 + ... + 2^{i-1} = 2^k - 1$$

Chuỗi $1 + a + a^2 + ... + a^n + ...$ khi a = 2 có tổng là: $\frac{1-2^k}{1-2}$

vì $$S_n = \frac{1-a^{n+1}}{1-a} \quad \text{khi } a \neq 1$$

Trong ví dụ trên với k = 6 ta có số nút tối đa là $2^6 - 1 = 63$

- Nếu một *cây nhị phân đầy đủ* mức k được gọi là cây nhị phân k_mức có 2^k-1. (Ngược lại, ta thấy, một cây nhị phân đầy đủ N nút thì có mức là $[\log_2 N] + 1$). Trong cây nhị phân đầy đủ các nút được đánh số theo trình tự từ trên xuống dưới, từ trái qua phải. Cách đánh số này cho ta định nghĩa một cây nhị phân hoàn thiện. Một cây nhị phân N nút với k mức là hoàn thiện khi và chỉ khi các nút của nó tương đương với việc đánh số từ 1 đến n trong cây nhị phân đầy đủ mức k. Chúng ta có thể lưu các nút trong mảng *tree* một chiều, nút thứ i sẽ lưu trong phần tử *tree[i]*.

Ta có bổ đề sau:

Nếu một cây nhị phân hoàn thiện N nút được biểu diễn tuần tự như trên thì với một nút i bất kỳ ta có:

(i) nút bố (i) sẽ ở vị trí (i div 2) với $i \neq 1$. Khi i = 1, i sẽ là gốc và không có nút bố.

(ii) nút con trái (i) sẽ ở vị trí 2i nếu $2i \leq n$. Nếu $2i > n$ thì i không có nút con trái

(iii) nút con phải (i) sẽ ở vị trí 2i+1 nếu $2i+1 \leq n$. Nếu $2i+1 > n$ thì i không có nút con phải.

Chứng minh:

Ta chứng minh cho (ii), còn (iii) sẽ suy ra từ (ii) và từ việc đánh số nút trên cùng một mức từ trái sang phải. (i) sẽ suy ra từ (ii) và (iii).

Chứng minh theo quy nạp: Với i = 1, nút con trái rõ ràng là bằng 2, và khi $2 > n$ thì 1 không có nút con trái.

Giả thiết là với mọi j, $1 \leq j \leq i$, nút con trái j ở vị trí 2j. Khi đó hai nút ở ngay phía trước nút trái (i-1) sẽ là nút phải của i và nút trái của i. Nút con trái của i là 2i. Như vậy, nút con trái của i + 1 là $2i + 2 = 2(i + 1)$ trừ khi trong trường hợp $2(i+1) > n$ thì i+1 sẽ không có nút con trái.

3.8.5. Định nghĩa đệ quy cây nhị phân và duyệt cây

Một cây nhị phân:

- *Hoặc là rỗng (phần neo)*

hoặc

- *Chứa một nút gọi là gốc, nó có hai con trỏ chỉ đến hai cây con nhị phân bên trái, bên phải.*

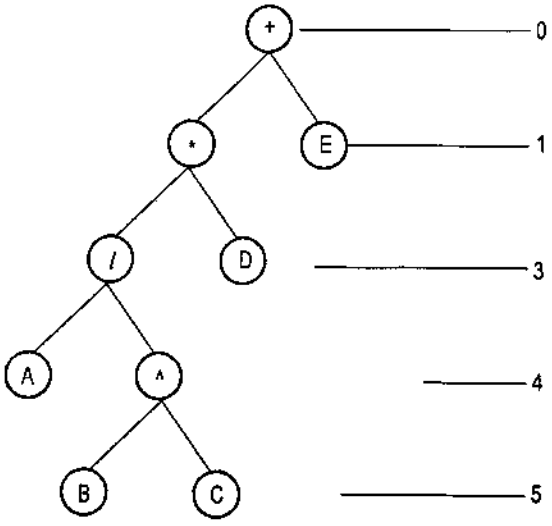
Việc duyệt cây đệ quy có ba bước cơ bản:

1. Xử lý nút gốc (D - Du_Lieu)
2. Duyệt cây con bên trái (L - moving left)
3. Duyệt cây con bên phải (R-moving right)

Ta có thể có 6 thứ tự duyệt:

LDR, LRD, DLR, DRL, RDL, RLD

Nếu ta chỉ thừa nhận chỉ duyệt từ trái sang phải thì chỉ có ba thứ tự duyệt chính là: LDR, LRD, DLR và thường được gọi là Inorder, Postorder, Preorder. Xét cây:



Ta xét ba kiểu duyệt nút:

a) LDR: Với kiểu duyệt này, ta đi xuống bên trái của cây cho đến khi không đi tiếp được nữa. Sau đó "thăm" nút, chuyển lên "thăm" nút gốc con, sau đó xuống bên phải và tiếp tục. Nếu không chuyển được xuống phải nữa thì "thăm" nút rồi quay trở lên một nút.

Gọi Inorder	*Giá trị ở gốc*	*Kết quả*
gốc chính	+	
1	*	
2	/	
3	A	
4	Nil	viết 'A'
4	Nil	viết '/'
3	^	
4	B	
5	Nil	B
5	Nil	^
4	C	
3	Nil	C
3	Nil	*

2	D	
3	Nil	D
3	Nil	+
1	E	
2	Nil	E
2	Nil	

Kết quả: A/B^C*D+E

```
Procedure  Inorder(Goc: TreePointer); { LDR}
        begin
            if  Goc < > Nil  then
            begin
                Inorder (Goc ^. Lchild);
                write(Goc ^.Du_Lieu);
                Inorder(Goc^.Rchild)
            end (*if*)
        end; (* of Inorder)
```

b) DLR (sẽ cho kết quả + */A** BCDE)

```
    Procedure Preorcler(Goc: treePointer); {DLR}
      begin  If goc < > Nil then
            begin
              write ( Goc ^.Du_Lieu).
              Preorder(Goc^. Lchild);
              Preorder ( Goc^. Rchild);
            end
      end;
```

c) LRD

```
Procedure Postorder (Goc: treePointer);
```

```
begin  if Goc < > Nil then
          begin  Postorder (Goc^.Lchild);
                 Postorder (goc^.Rchild);
                 write (goc^,Du_Lieu);
          end {of if}
end;
```

Sẽ cho kết quả: ABC ** /D*E+

3.9. BÀI TẬP

3.9.1. Bài tập phần bản ghi

1. Định nghĩa bản ghi, truy nhập bản ghi?
2. Câu lệnh with, cho ví dụ và giải thích.
3. Thế nào là bản ghi có cấu trúc thay đổi?

3.9.2. Bài tập vẽ kiểu dữ liệu tập hợp

1. Thế nào là kiểu dữ liệu tập hợp?
2. Các tập hợp bằng sau có bằng nhau?

   [1, 2, 3, 4, 4, 5]; [2, 3, 5, 4, 1]; [4, 1, 3, 2, 5]

3. Viết chương trình kiểm tra ký tự nhập vào có phải là chữ cái không ?

3.9.3. Bài tập vẽ danh sách liên kết

1.Viết chương trình đọc các bản ghi trong tệp dưới dạng một danh sách liên kết có dùng con trỏ

Thứ tự	Họ đệm	Tên	Điểm trung bình	Học bổng
1 N2	C18	C7	N4.1	N6.1
20				

2. Dùng con trỏ viết chương trình quản lý thư viện:

Ký hiệu	Tên sách	Tác giả	Năm xuất bản	Nhà xuất bản

a) Nhập dữ liệu

b) Thêm bản ghi mới

c) Xoá bỏ bản ghi

d) Tìm sách theo ký hiệu

3.9.4. Bài tập vẽ hàng đợi liên kết

Viết các thủ tục/hàm để cài đặt hàng đợi liên kết trên cơ sở con trỏ:

1. CreateQ - Tạo hàng đợi
2. EmptyQ - Kiểm tra hàng đợi rỗng
3. AddQ - Thêm phần tử
4. RemoveQ - Xóa phần tử

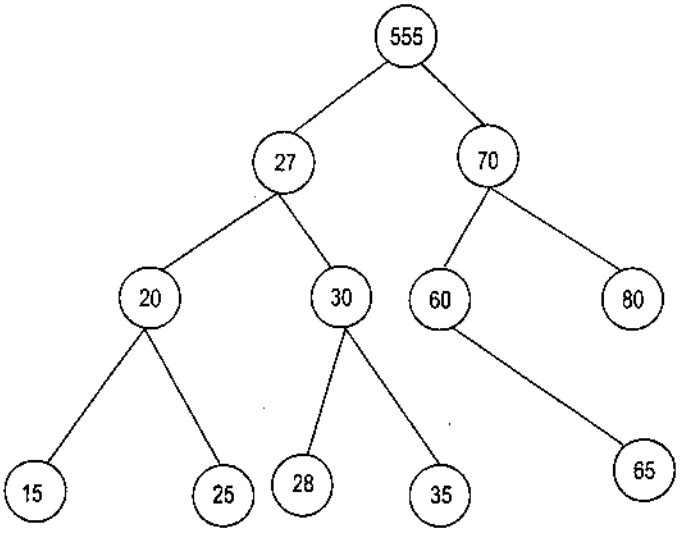
3.9.5. Bài tập vẽ đệ quy

1. Viết hàm đệ quy tính USCLN của hai số nguyên.
2. Viết thủ tục đệ quy (hàm) tính hệ số nhị thức

$$C_{nk}^{n} = \frac{N!}{K!(N-K)!}$$

3.9.6. Bài tập vẽ cây nhị phân

1. Cho cây nhị phân:



Hãy hiển thị kết quả tạo bởi kiểu duyệt

- Inorder (LDR)
- Preorder (DLR)
- PostOrder (LRD)

2. Với mỗi biểu thức số học dưới đây, hãy vẽ cây nhị phân biểu diễn biểu thức ấy rồi dùng các kiểu duyệt để tìm biểu thức tiền tố và hậu tố tương đương.

a) (A-B)-C

b) A- (B-C)

c) A/(B-(C-(D-(E-F))))

d) ((((A-B)-C)-D)-E)/F

e) ((A*(B + C))/(D - E + F)))*(G/(H/(I*J)))

3. Kiểu duyệt tiền tự tạo ra ADFG HKLPQRWZ

Kiểu duyệt trung tự tạo ra GFHKDLAWRQPZ

Hãy vẽ cây nhị phân.

Phần II
THUẬT TOÁN

Chương 4
NHẬP MÔN THUẬT TOÁN

4.1. XUẤT XỨ CỦA THUẬT TOÁN

Thuật ngữ "Thuật toán" có xuất xứ từ tên nhà Toán học người Thổ Nhĩ Kỳ sống ở thế kỷ thứ 9 Abu Abdullah Muhammad ibn Musa Al-Khwarizmi. Nhà toán học Al-Khwarizmi có viết quyển sách về số học mà đến thế kỷ XVIII được truyền sang đến châu Âu, về sau, người ta đã gọi tên tác giả (Al-Khwarizmi) kèm với các phương pháp số học (Arithmetic) theo phiên âm tiếng Latin là Algorithm - Thuật toán - để chỉ các thủ tục, các phương pháp giải quyết vấn đề hay các nhiệm vụ tính toán.

Abu Abdullah Muhammad ibn Musa al-Khwarizmi

Abu Abdullah Muhammad bin Musa al-Khwarizmi vừa là nhà toán học, thiên văn học, chiêm tinh học và địa chất học. Ông sinh khoảng năm 780 ở Khwarizm và mất khoảng năm 850. Sau khi Đạo Hồi đã thống trị vùng Ba Tư, Baghdad trở thành một trung tâm nghiên cứu khoa học và thương mại, đã có rất nhiều thương gia và các nhà khoa học từ các miền xa xôi như Trung Quốc và Ấn Độ cũng

đã đến đây lập nghiệp. Abu Abdullah Muhammad bin Musa al-Khwarizmi cũng đã sống và làm việc ở đây với tư cách Nhà học giả của Viện Trí thức (House of Wisdom), ông chuyên nghiên cứu và dịch các tài liệu khoa học từ tiếng Hy Lạp.

Sách của Ông là cuốn sách đầu tiên giải quyết một cách hệ thống về giải các phương trình bậc nhất và bậc hai. Chính thuật ngữ Đại số (algebra) cũng có xuất xứ từ thuật ngữ al-jabr trong các tài liệu gốc của Ông.

Sự phát triển của thuật toán gắn liền với sự phát triển của các công cụ tính toán, sự phát triển của Công nghệ thông tin.

Trong số các công cụ người Cổ đại dùng để thể hiện cách giải các bài toán của mình phải kể đến các hòn sỏi, các cây que. Người Cổ đại đã dùng chúng để đếm, tính toán các cuộn len, các bao tải ngũ cốc hay tiền vàng. Các loại đánh dấu và ký hiệu này có tên gọi là ký kiệu Tally. Các ký hiệu Tally đã đóng góp một phần trong hệ thống số học được sử dụng trong Máy Turing và hệ thống tính toán Máy Post - Turing.

Những sáng chế tiếp theo vào thời Trung đại là các đồng hồ cơ khí. Việc chế tạo ra các đồng hồ chính xác đã dẫn đến các "máy tự động chính xác" ở thế kỷ XVIII và cuối cùng là các máy tính của Charle Babbage và Countess Ada Lovelace.

Điện tín, điện thoại (năm 1835) và Bìa đục lỗ Hollerith (1890) là những bước tiến kế tiếp về kỹ thuật và cơ khí hóa tính toán. Những năm cuối thế kỷ XIX và nửa đầu thế kỷ XX, Toán học đã có nhiều đóng góp to lớn cho phát triển kỹ thuật tính toán. Trước hết phải kể đến các công trình của George Boole (1847) về Đại số Boole, về các nguyên tắc toán học biểu diễn trong ngôn ngữ ký hiệu của Giuseppe Peano (1888).

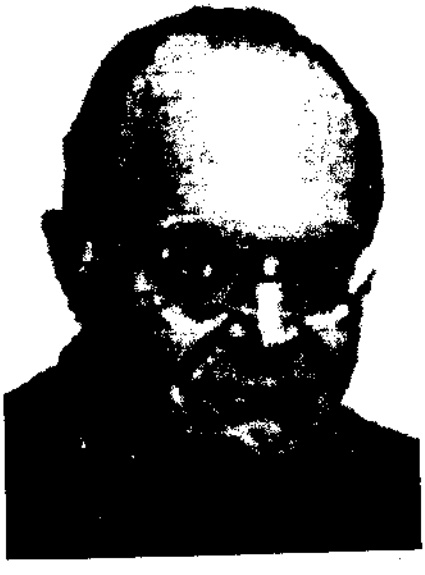
Việc thể hiện các thuật toán tính toán bằng máy tính được các nhà toán học lỗi lạc tiếp theo mô tả trong những công trình nổi tiếng - Máy Turing (1936-1937) do nhà tin học Alan Turing đề xướng. Độc lập với Turing, Emil Post cũng mô tả một quá trình tương tự. Về sau người ta gọi chung tên hai nhà bác học này là Máy Post-Turing.

Những định nghĩa mới về thuật toán tính toán trên máy tính điện tử được các nhà Tin học nổi tiếng khác đề xướng, đó là J Barkley Rosser (1939) và Stephen C.Kleene (1943). Theo ý nghĩa Toán học và Tin học,

một thuật toán chính là một thủ tục bao gồm một dãy các bước tính toán xử lý đã được xác định rõ ràng để thực hiện các nhiệm vụ tính toán, cho những kết quả xác định trong một khoảng thời gian xác định. Mức độ phức tạp của thuật toán, sự ứng dụng có hiệu quả của nó là những yếu tố quan trọng, và một phần đóng góp không nhỏ vào sự thành công này là việc cấu trúc dữ liệu thích hợp.

Thuật toán chính là các chỉ thị cách thức để máy tính xử lý thông tin, vì một chương trình máy tính chính là một thuật toán, chỉ định cho máy tính thực hiện từng bước phải xử lý cái gì trước, cái gì sau, phải in kết quả nào ra giấy hay ra đĩa v.v...

Một trong những người có đóng góp nhiều cho thuật toán trong Công nghệ thông tin là Donald Ervin Knut. Ông sinh ngày 10 tháng Giêng năm 1938 tại Milwaukee, Wisconsin, Mỹ.



D. E. Knut

Từ năm 1958, D.E.Knut đã viết chương trình máy tính phân tích hoạt động của đội bóng basketball của Trường trung cấp. Năm 1968, Ông cho ra mắt công trình đồ sộ "Nghệ thuật lập trình máy tính - The Art of Computer Programming", tập 1 "Các thuật toán cơ bản - Fundamental Algorithms". D.E.Knut là tác giả của thuật toán Knuth-Bendix- là thuật toán nền tảng tính toán các cấu trúc đại số, đặc biệt cho cấu trúc nhóm và nửa - nhóm. Ông cũng là tác giả của phần mềm TeX - là ngôn ngữ dùng cho soạn thảo các tài liệu toán học và khoa học, tác giả của METAFONT - phần mềm hệ thống để thiết kế ký font.

Khi nghiên cứu về thuật toán, D.E.Knut đã liệt kê ra 5 tính chất cần có đối với một thuật toán: Có thời gian thực hiện hữu hạn, có số bước thực hiện hữu hạn; Mỗi bước của thuật toán phải hoàn toán xác định; Dữ liệu ban đầu phải được xác định trước; Lượng kết quả kết xuất; Tính hiệu quả xét về mặt thời gian và độ chính xác của kết quả tính toán.

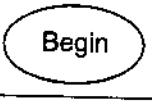
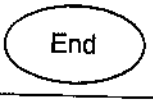
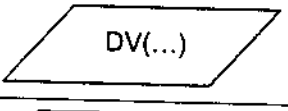
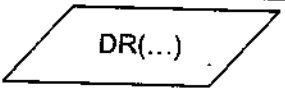
4.2. CÁC PHƯƠNG PHÁP TRÌNH BÀY THUẬT TOÁN

Thuật toán là mẹo luật, là các bước của quá trình xử lý tin. Chương này sẽ trình bày thuật toán theo tư tưởng lập trình có cấu trúc, dưới góc độ của người lập trình. Trong các tài liệu tin học, hai thuật ngữ *thuật toán* và *thuật giải* là như nhau. Một thuật toán tốt phải đảm bảo được yêu cầu: tính đầy đủ, tổng quát, chặt chẽ về mặt lôgic.

4.2.1. Phương pháp sơ đồ khối

4.2.1.1. Các ký hiệu trong sơ đồ khối

Trong sơ đồ khối sử dụng các khối và mũi tên chỉ hướng, trình tự thực hiện các phép toán. Khi vẽ sinh viên phải vẽ đúng quy định:

Ký hiệu	Ý nghĩa
	Bắt đầu sơ đồ
	Kết thúc sơ đồ
	Nhập dữ liệu
	Xuất kết quả
	Tính toán, xử lý
	So sánh, kiểm tra
	Ký hiệu chương trình con
	Ký hiệu tiếp tục, mốc nối

4.2.1.2. Ví dụ

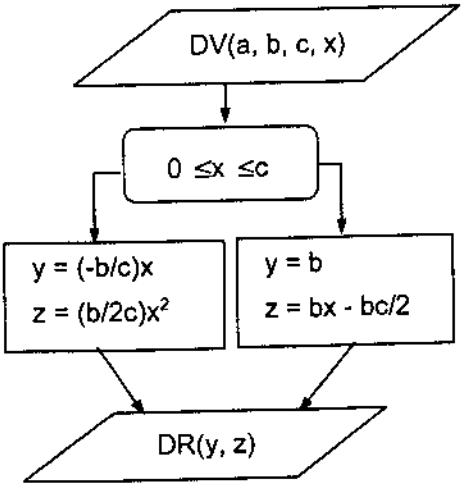
Lập sơ đồ khối tính tung độ y và diện tích z của phần gạch chéo ở hình 4.1, phụ thuộc vào giá trị x khi điểm M di chuyển trên AB. Cho biết AB = a, BC = b, AE = c.

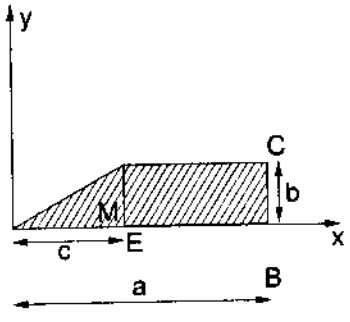
Từ hình vẽ ta thấy:

$$y = \begin{cases} (-b/c)x & \text{nếu } 0 \le x \le c \\ b & \text{nếu } c < x \le a \end{cases}$$

$$z = \begin{cases} (b/2c)x^2 & \text{nếu } 0 \le x \le c \\ bx - bc/2 & \text{nếu } c < x \le a \end{cases}$$

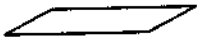
Thuật toán bài trên có dạng:

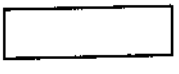


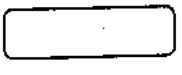


Hình 4.1

Trong sơ đồ này sử dụng cả ba loại khối:

Khối vào/ra 

Khối tính toán 

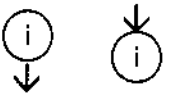
Khối so sánh 

Khi lập sơ đồ khối phải lưu ý:

- Những đại lượng tham gia tính toán, được cho giá trị thì phải đưa vào khối đưa vào, khối đưa ra chứa tên những đại lượng hoặc dòng thông báo sẽ đưa ra màn hình hay máy in, là kết quả của bài toán.

- Trên sơ đồ, khối kiểm tra chỉ có một lối vào và hai lối ra, ứng với điều kiện kiểm tra đúng (dấu +) và sai (–). Sau này ta sẽ xét kiểm tra theo nhiều trường hợp, có nhiều lối ra hơn.

- Các câu lệnh tính toán được để trong các khối chữ nhật. Sơ đồ khối cho phép ta theo dõi thuật toán một cách trực giác, đơn giản. Nhược điểm của nó là cồng kềnh. Nếu sơ đồ khối kéo dài trên nhiều trang, ta phải vẽ các khớp nối các khối:



4.2.2. Phương pháp sơ đồ tuyến

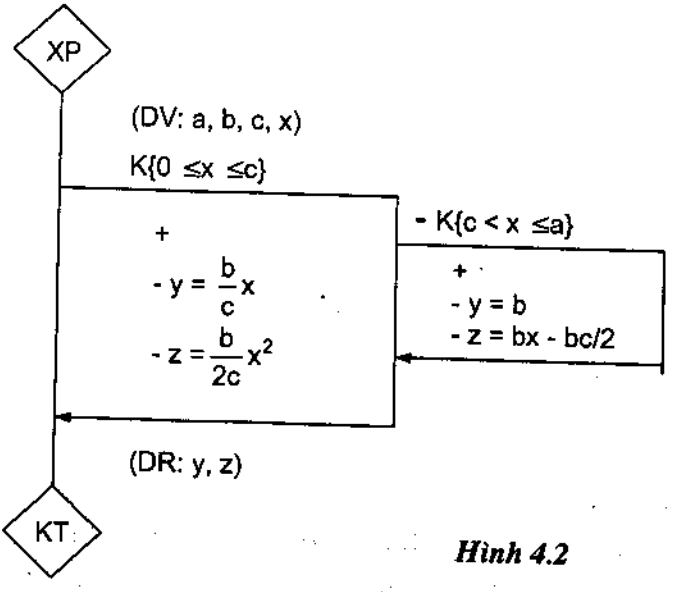
4.2.2.1. Khái niệm, quy định

Ngoài sơ đồ khối, ta có thể dùng các cách biểu diễn khác như sơ đồ tuyến, sơ đồ toán tử, cách phác thảo chương trình...

Sơ đồ tuyến là dạng sơ đồ được triển khai trên một trục dọc, các phép tính được trình bày bên phải của trục.

4.2.2.2. Ví dụ

Sơ đồ tuyến của bài toán trên như sau:



Hình 4.2

Ở đây, K {...} là kiểm tra điều kiện.

4.2.3. Phương pháp phác thảo

Để làm ví dụ cho phương pháp phác thảo ta xét bài toán tìm ước số chung lớn nhất (USCLN) của hai số x, y.

Ví dụ 4.2:

1. Đưa vào x, y;
2. x = |x| (trị tuyệt đối)
3. y = |y|;
4. z = x-y*([x/y])([...]) là lấy nguyên;
5. Nếu z = 0 suy ra y là USCLN, kết thúc;
6. Nếu z = 0 tiếp tục.
7. x = y, y = z;
8. Quay lại bước 4.

4.2.4. So sánh các phương pháp trình bày thuật toán

Các phương pháp trình bày thuật toán đều có ưu nhược điểm riêng.

Phương pháp sơ đồ khối có ưu điểm là dễ nhìn, trực quan, cho phép vẽ được các sơ đồ kéo dài trên nhiều trang với ký hiệu móc nối, nhược điểm của sơ đồ này là cồng kềnh.

Phương pháp tuyến có ưu điểm là gọn nhẹ, nhược điểm là khó theo dõi khi bài toán lớn, phức tạp.

Phương pháp phác thảo đơn giản, gần với cách viết thông thường, nhược điểm là bất tiện khi thể hiện vòng lặp.

4.3. BÀI TẬP

1. Dùng phương pháp sơ đồ khối, sơ đồ tuyến, cách phác thảo để trình bày thuật toán giải phương trình bậc hai $ax^2 + bx + c = 0$;

2. Dùng phương pháp sơ đồ khối, sơ đồ tuyến, cách phác thảo để trình bày thuật toán giải bài toán tách một dãy số nguyên thành 2 dãy chẵn lẻ.

Chương 5

CÁC DẠNG THUẬT TOÁN CƠ BẢN

5.1. THUẬT TOÁN CÓ CẤU TRÚC

Trong quá trình phát triển tin học, nghệ thuật lập trình không ngừng được cải tiến, nhiều phương pháp lập trình mới được đề nghị như thiết kế thuật toán và chương trình theo môđun, lập trình có cấu trúc, cách tiếp cận từ trên xuống (top - down), kỹ thuật đi từ dưới lên (bottm - up) v.v... Khi giải quyết các bài toán lớn, xây dựng thuật toán cho nó thường phải kết hợp đồng thời các kỹ thuật trên.

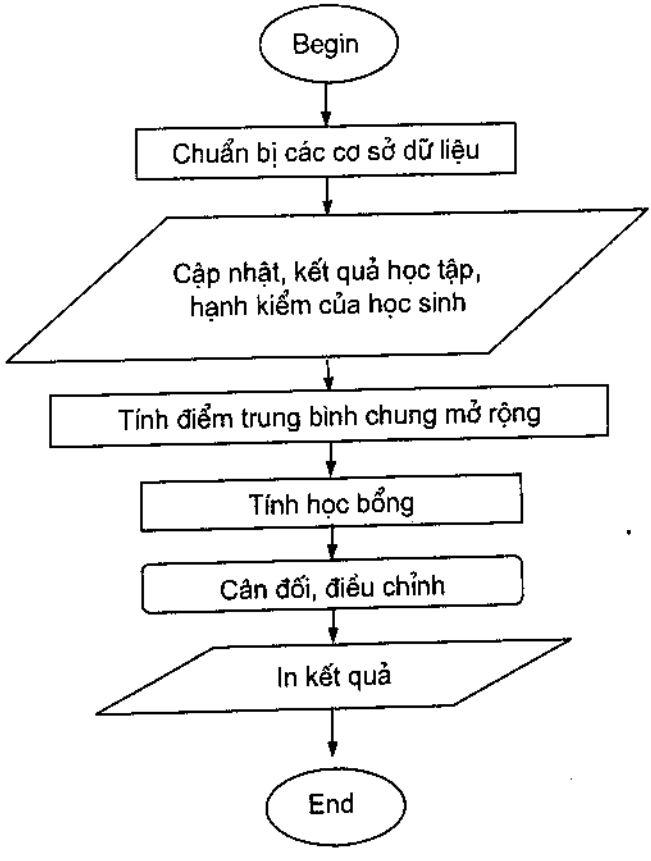
5.1.1. Thiết kế theo môđun

Với những bài toán lớn, hệ thống lớn, khi xây dựng thuật toán ta thường phải chia nhỏ ra thành từng khối, mỗi khối thực hiện một số nhiệm vụ xử lý nhất định. Khi chuyển thành chương trình, mỗi khối này sẽ là một môđun. Đó chính là tư tưởng của kỹ thuật lập trình theo môđun. Mỗi môđun có những đặc trưng sau:

- Thực hiện trọn gói một số nhiệm vụ xử lý tin cụ thể;
- Chỉ có một lối vào và một lối ra;
- Có thể ghép nối vào trong quá trình xử lý tin khác mà không cần thay đổi gì trừ thay đổi dữ liệu ban đầu;
- Có thể gọi, sử dụng các môđun khác.

Lập trình theo môđun có lợi ở chỗ các môđun có thể giao cho các nhóm lập trình, sau đó ghép lại trong một chương trình quản lý chung, mỗi môđun có độ phức tạp vừa phải cho phép dễ theo dõi, sửa chữa, bổ sung.

Ví dụ 5.1: Sử dụng kỹ thuật thiết kế môđun để xây dựng chương trình quản lý đào tạo. Đây là một bài toán khá phức tạp, bao gồm nhiều phần việc, chúng được tổ chức theo các môđun trong sơ đồ khối như trên hình 5.1.



Hình 5.1

Lập trình theo môđun cho phép tổ chức tính toán một cách tối ưu. Các nhiệm vụ tính toán, xử lý khác nhau được tổ chức thành các môđun khác nhau, mỗi môđun là một chương trình con, khi cần tính toán hay xử lý chỉ việc gọi nó. Chúng ta cũng có thể sửa đổi, bổ sung các môđun mà không ảnh hưởng đến toàn bộ chương trình lớn.

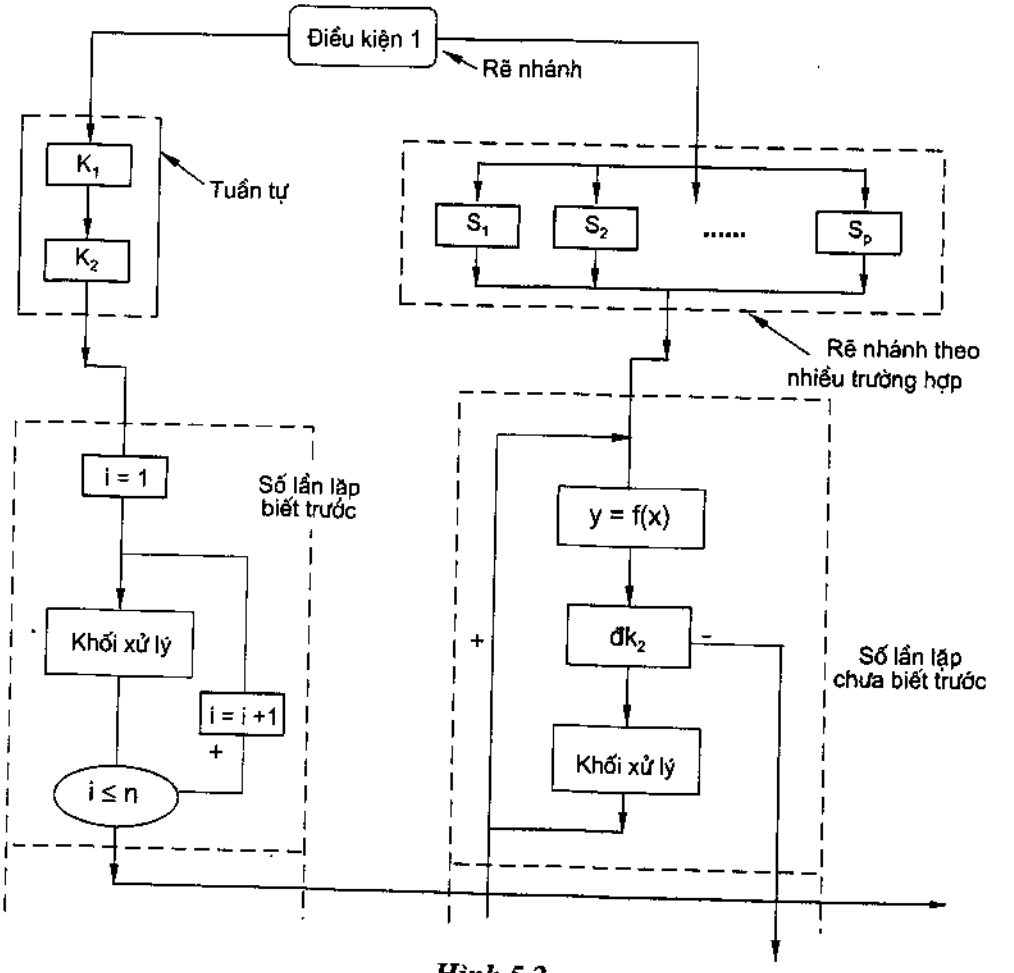
5.1.2. Thuật toán có cấu trúc

Mỗi môđun thuật toán thường gồm nhiều thủ tục tính toán, xử lý. Diễn đạt các thủ tục xử lý tin một cách rõ ràng, lôgic, dễ phát hiện sai sót, dễ khẳng định tính đúng đắn của thuật toán là điều quan trọng trước khi viết chương trình. Dijkstra E.W đã đề xướng một phương pháp hữu hiệu, giúp cho vẽ thuật toán có hiệu quả, đó là phương pháp lập trình cấu trúc. Theo quan điểm lập trình cấu trúc, các thuật toán được phân ra làm 4 dạng cấu trúc cơ bản: Tuần tự, rẽ nhánh đơn giản, rẽ nhánh theo nhiều trường hợp,

xử lý lặp hay thường gọi là thuật toán chu trình; thuật toán của bài toán lớn được chia thành nhiều khối thuộc một trong bốn cấu trúc cơ bản trên và chỉ có một lối vào, một lối ra.

Thuật toán gồm các cấu trúc như vậy gọi là thuật toán có cấu trúc, chương trình tương ứng gọi là chương trình có cấu trúc.

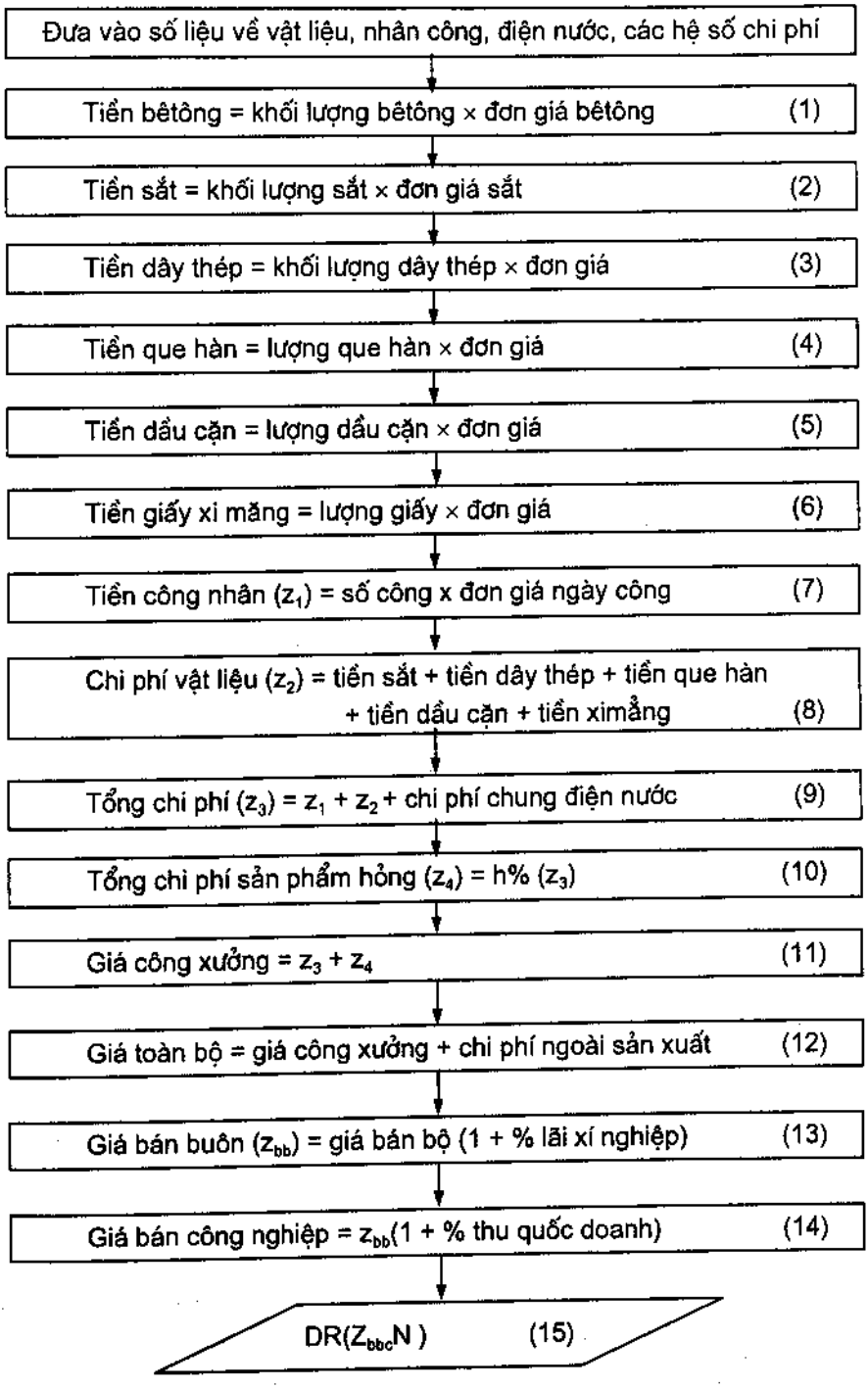
Hình 5.2 giới thiệu một môđun thuật toán có chứa các cấu trúc cơ bản.



Hình 5.2

Để cho mỗi khối có tính độc lập, trong nó phải có đủ ba thành phần: chuẩn bị dữ liệu ban đầu, phần tính toán, phần kết thúc khối. Lập trình cấu trúc hạn chế tối đa việc chuyển điều khiển đến các mốc (nhãn), chỗ nào có nhãn phải thay bằng một khối cấu trúc. Sau đây sẽ xét các dạng thuật toán cơ bản đã nêu.

5.2. THUẬT TOÁN CỦA BÀI TOÁN TUẦN TỰ



Hình 5.3

Ví dụ 5.2: Giá bán một sản phẩm bêtông cấu thành từ các khoản mục chi phí: - bêtông; - sắt; - que hàn; - dây thép;- giấy ximăng; - dầu cặn; - điện nước; - chi phí sản phẩm hỏng; - chi phí ngoài sản xuất(%); - chi phí công xưởng(%); - nộp thu quốc doanh(%); - nhân công;

Mỗi loại sản phẩm bêtông có một con số riêng về chi phí từng loại vật liệu (số lượng, chủng loại, giá cả) về điện nước, về các loại chi phí như nhân công,... Tính giá cho một sản phẩm bêtông là tính chi phí cụ thể cho từng khoản mục sau đó cộng lại.

Ta có sơ đồ như trên hình 5.3. Trong sơ đồ, quá trình tính toán đi tuần tự từ khối đưa dữ liệu vào, qua các khối số 1, 2... cho đến 15, không phải rẽ nhánh, không tính lặp. Dạng sơ đồ tuần tự như vậy còn được gọi là thuật toán đơn giản nhất.

5.3. THUẬT TOÁN RẼ NHÁNH ĐƠN GIẢN

Dạng tuần tự thường chỉ là một đoạn, một bước nào đó trong cả bài toán, còn lại phần lớn là thuật toán đều chứa một hay nhiều phần rẽ nhánh. Sơ đồ khối trong hình 2.4 cho ta một hình ảnh rõ ràng về sự rẽ nhánh. Trong đó, nếu $0 \le x \le c$ đúng (+) thì quá trình tính toán sẽ theo phía trái và ngược lại (-), rẽ sang phải.

Ví dụ 5.3: Hệ số tăng hiệu quả sử dụng vốn đầu tư vào thiết bị được tính theo công thức:

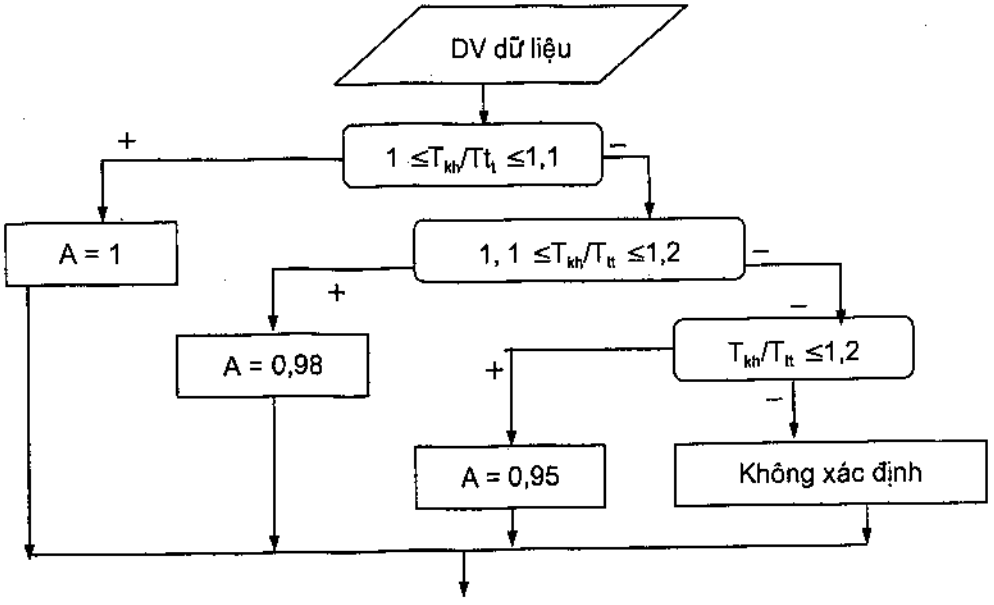
$$K_c = 1 + A\frac{T_{kh} - T_{tt}}{T_{tt}} + L(K_{lkh} - K_{ltt}) - \frac{M_t - M_{tt}}{M_t}$$

Trong đó hệ số A sẽ nhận giá trị theo điều kiện sau:

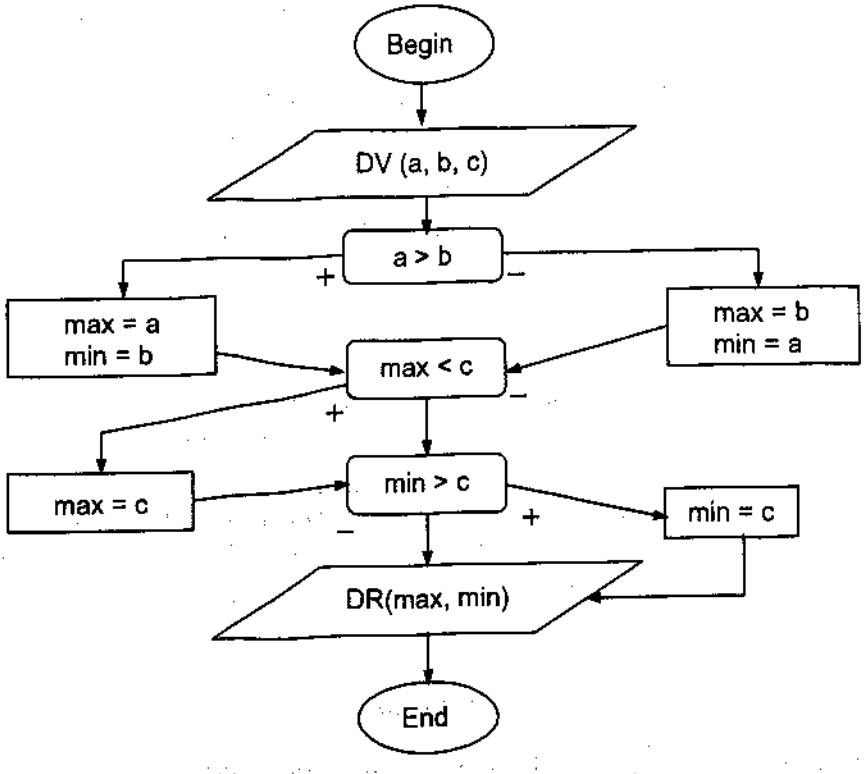
$$A = \begin{cases} 1 & \text{nếu} \quad 1 \le T_{kh}/T_{tt} \le 1,1 \\ 0,98 & \text{nếu} \quad 1,1 \le T_{kh}/T_{tt} \le 1,2 \\ 0,95 & \text{nếu} \quad T_{kh}/T_{tt} > 1,2 \end{cases}$$

Ta có sơ đồ như trên hình 5.4.

Trong ví dụ trên, vì A nhận từng giá trị phụ thuộc vào từng khoản giá trị của T_{kh}/T_{tt} nên ta phải tiến hành các khối kiểm tra cụ thể, chứ không thể kiểm tra một lần. Nếu trong bài toán chỉ có hai khả năng rẽ nhánh thì phải kiểm tra một lần.



Hình 5.4



Hình 5.5

Ví dụ 5.4: Cho ba số a, b, c lập sơ đồ thuật toán chọn số nhỏ nhất và lớn nhất của chúng. Ta ký hiệu max là biến sẽ nhận giá trị lớn nhất, min-bé nhất. Trước hết, so sánh a và b, nếu a>b thì max = a, min = b, ngược lại thì max = b, min = a. Sau đó ta so sánh max, min với c để chọn ra số lớn nhất và số nhỏ nhất sơ đồ có dạng như trên hình 5.5.

5.4. THUẬT TOÁN RẼ NHÁNH THEO NHIỀU TRƯỜNG HỢP

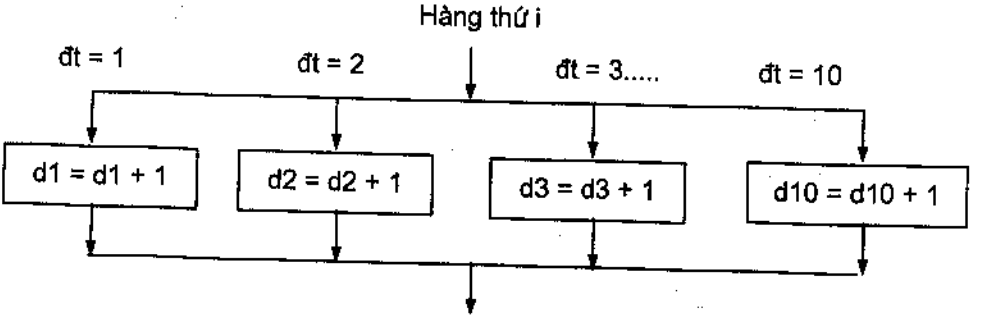
Ví dụ 5.5: Có mười đối tượng thí sinh dự thi, mỗi thí sinh chỉ đăng ký một đối tượng:

TT	Họ và tên	Đối tượng
1	H.B.A	9
2	H.T.Q.A	2

Lập sơ đồ khối để thống kê mỗi loại đối tượng có bao nhiêu thí sinh. Để vẽ thuật toán rẽ nhánh theo nhiều trường hợp cho ví dụ này, ta làm như sau:

- Đặt biến dt (đối tượng) để kiểm tra rẽ nhánh; Tạo 10 bộ đếm d1 = 0, ..., d10 = 0 (lúc đầu gán = 0) để mỗi lần so sánh giá trị trong cột đối tượng, sẽ tăng dk lên một đơn vị nếu dt bằng k.

Tại mỗi dòng (mỗi thí sinh) phải thực hiện công việc kiểm tra và rẽ nhánh theo trường hợp như trên hình 5.6.



Hình 5.6

Trong ngôn ngữ lập trình Pascal, rẽ nhánh theo nhiều trường hợp được thực hiện bởi câu lệnh case... of.

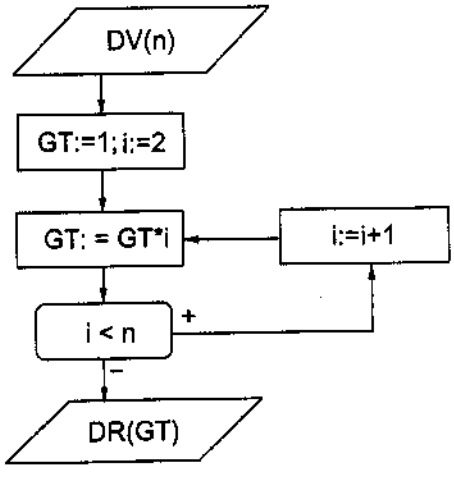
5.5. THUẬT TOÁN BÀI TOÁN LẶP (CHU TRÌNH)

Ta có thể chia bài toán lặp làm hai loại: có số lần lặp biết trước và có số lần lặp chưa biết trước. Trong mỗi loại đó lại chia ra các dạng: dạng tính tổng, dạng tính tích, dạng tính khác tổng và tích.

5.5.1. Lặp có số lần lặp biết trước

5.5.1.1. Bài toán tính tích

Ví dụ 5.6: Tính giai thừa của n : n! = 1.2.3.... (n-1)n. Phân tích bài toán này ta thấy, nếu ban đầu ta đặt GT = 1 sau đó sẽ tính lặp công thức GT = GT*i, bắt đầu i = 2 cho đến n thì kết quả sẽ cho n!. sơ đồ khối như ở hình 5.7 (*để bớt phức tạp, trong các sơ đồ không vẽ khối Begin và End*).



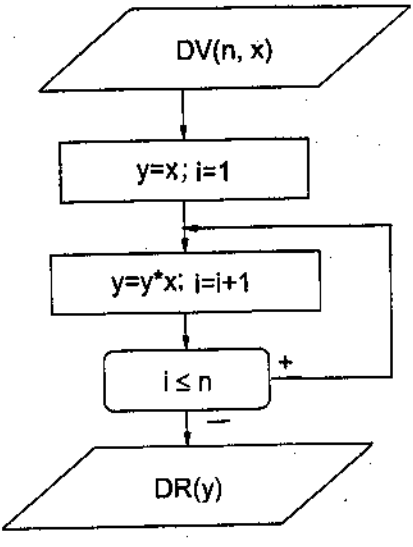
Hình 5.7

Đặc điểm của sơ đồ lặp tính tích là giá trị ban đầu của tích được gán bằng 1. Trong sơ đồ tính lặp, phải xác định được công thức tính toán chính mà sẽ được tính lặp (ở đây là GT = GT*i); biến i ở đây gọi là tham số điều khiển vòng lặp; (i < n) là kiểm tra điều kiện kết thúc lặp.

Ví dụ 5.7:

Tính $y = x^n$ hay $y = \underbrace{x.x.....x}_{n}$

Ta tổ chức như sau: có 4 đại lượng tham gia giá trình tính: y; x; n và một số đếm chu trình i, lúc đầu gán x cho y, (y: = x); i nhận giá trị từ hai (i: = 2) sau đó sẽ tính lặp công thức y: = y*x. Mỗi lần tính công thức này, lại tăng bộ đếm i lên một



Hình 5.8

đơn vị (i: = i+1) quá trình này lặp cho đến khi i>n. Sơ đồ khối cho trên hình 5.8.

5.5.1.2. Bài toán tính tổng

Ví dụ 5.8: Lập sơ đồ thuật toán tính tổng dãy số thực:

$$S = \sum_{i=1}^{n} a_i$$

Bài toán trên được phân tích như sau:

$$S = \underbrace{a_1 + a_2}_{s_1} + a_3 + ... + a_{n-1} + a_n$$

$$\underbrace{\quad}_{s_1} \quad \underbrace{\quad}_{s_1} .....$$

Lúc đầu S = 0.

Khi i = 1, $S = S + a_i = a_1$

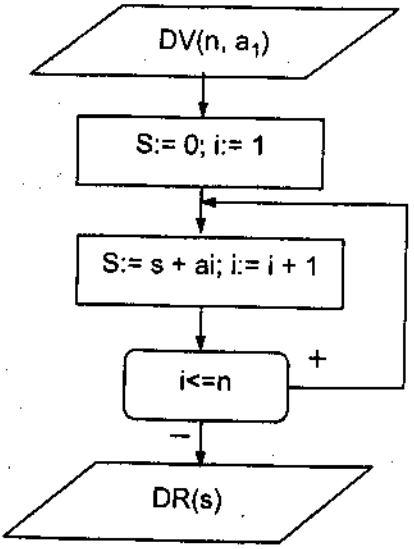
i = 2, $S = S + a_i = a_1 + a_2$

.....

i = n, $S = S + a_i = a_1 + a_2 + ... + a_n$.

Công thức tính lặp là $S = S + a_i$

Sơ đồ khối như trên hình 5.9.



Hình 5.9

Đặc điểm của sơ đồ thuật toán tính tổng là giá trị ban đầu của tổng gán bằng 0 (hoặc có thể gán giá trị ban đầu của tổng bằng phần tử đầu tiên của dãy). Các phần khác cũng giống sơ đồ tính tích.

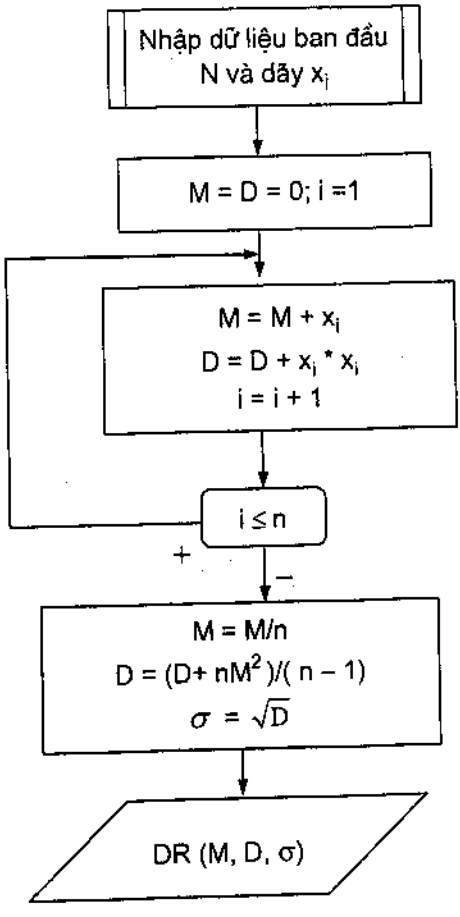
Ví dụ 5.9: Cho n giá trị thống kê, lập sơ đồ thuật toán tính các giá trị:

- Kỳ vọng toán học: $M = \frac{1}{n}\sum_{i=1}^{n} x_i$;

- Phương sai: $D = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - M)^2$

- Độ lệch trung bình bình phương: $\sigma = \sqrt{D}$.

Sơ đồ thuật toán trên hình 5.10.



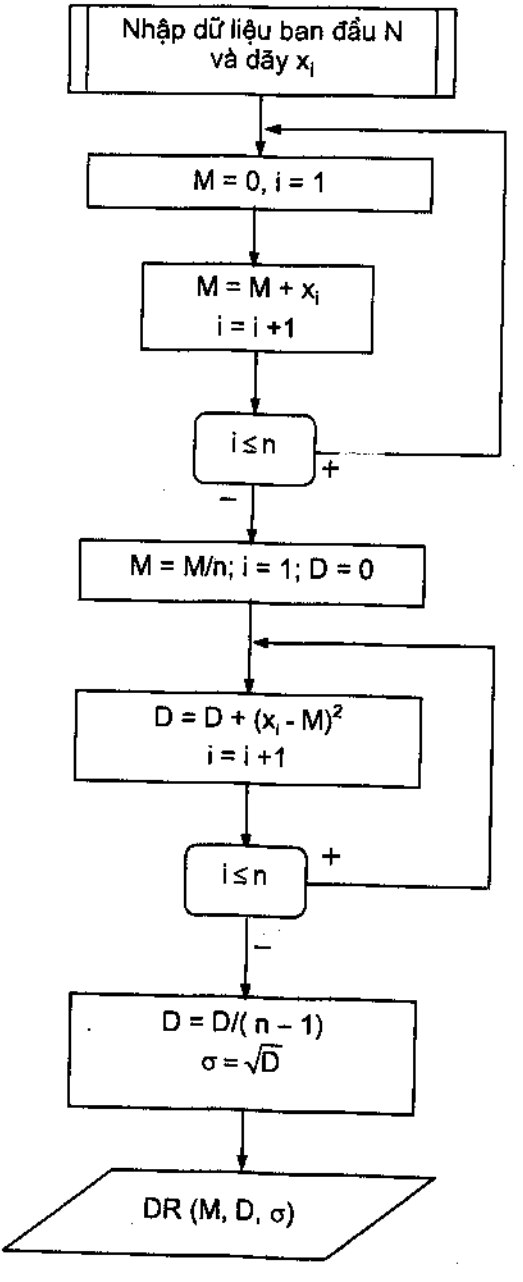
Hình 5.10

Sơ đồ trên có thể cải tiến hành sơ đồ đơn giản hơn (hình 5.11) bằng các biến đổi sau:

$$D = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - M)^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i^2 - 2x_iM + M^2)$$

$$= \frac{1}{n-1}\left[\sum_{i=1}^{n}x_i^2 - 2M\sum_{i=1}^{n}x_i + \sum_{i=1}^{n}M^2\right]$$

$$= \frac{1}{n-1}\left[\sum_{i=1}^{n}x_i^2 - 2nM^2 + nM^2\right] = \frac{1}{n-1}\left[\sum_{i=1}^{n}x_i^2 - nM^2\right]$$


Nhập dữ liệu ban đầu N
và dãy x_i
M = 0, i = 1
M = M + x_i
i = i +1
i≤n
+
–
M = M/n; i = 1; D = 0
D = D + (x_i - M)^2
i = i +1
i≤n
+
–
D = D/(n – 1)
σ = √D
DR (M, D, σ)

Hình 5.11

Ví dụ 5.10:

Tính tích phân xác suất:

$$p(x) = \frac{1}{\sqrt{2\pi}} \int_{-x}^{x} e^{-t^{2/2}} dt \qquad (5.1)$$

Trước hết, ta phân tích để lập sơ đồ thuật toán tính một tích phân định hạn tổng quát:

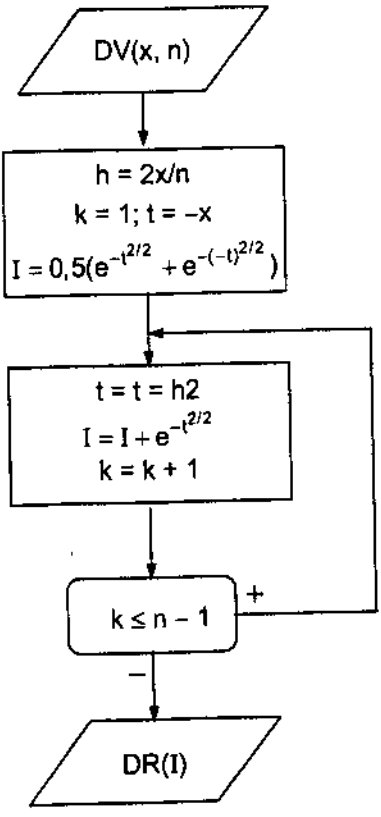
$$I = \int_{a}^{b} f(x)dx \qquad (5.2)$$

Từ toán giải tích, ta đã biết công thức:

$$I = \int_{a}^{b} f(x)dx = h \sum_{i=0}^{n-1} f(x_i); \qquad (5.3)$$

trong đó: h là bước chia

$h = (b - a)/n.$



Hình 5.12

Triển khai công thức (5.3) (tính tích phân theo phương pháp hình thang).

$$I = 0,5\left[f(a) + f(a+h)\right] + 0,5\left[f(a+h) + f(a+2h)\right] + \ldots + \left[a + (n-1)h + f(a)\right]h$$

Gộp những phần tử giống nhau ta có:

$$I = 0,5\left[f(a) + f(b) + h\sum_{i=1}^{n-1} f(a+ih) \right] \qquad (5.4)$$

Thay (5.4) vào (5.1) tb sẽ có công thức tính

$$P(x) = 0,5\left[(e^{-(-x)^{2/2}} + e^{-(x)^{2/2}}) + \sum_{i=1}^{n-1} e^{-(-x+ih)^{2/2}} \right] \frac{h}{\sqrt{2\pi}}$$

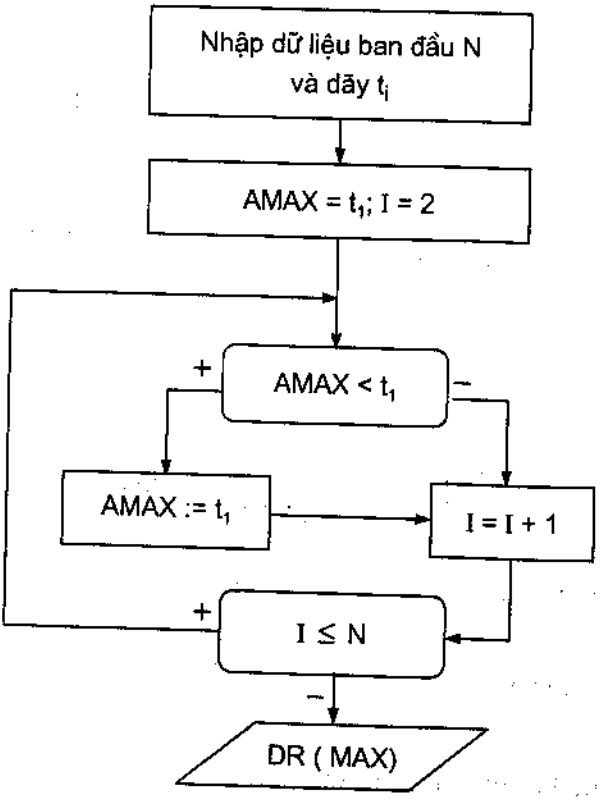
Sơ đồ khối của bài toán trên hình 5.12.

5.5.2. Thuật toán của bài toán lặp phân nhánh, lồng nhau

Ví dụ 5.11:

Lập sơ đồ thuận toán tính max(t_1, t_2,... t_i) với dãy t_1, t_2,... t_i cho trước. Để làm bài toán này ta phải sử dụng một biến AMAX để giữ lấy giá trị lớn nhất của dãy số t_1, t_2,... t_i. Lần thứ nhất ta gán t_1 cho AMAX, sau đó so sánh AMAX với t_2. Nếu AMAX bé hơn t_2, thì gán t_2 cho AMAX nếu không thì không làm gì cả, chuyển tiếp so sánh AMAX với t_3, v.v... Quá trình lặp lại cho đến khi đã duyệt hết dãy số. Sơ đồ thuật toán cho trên hình 5.13. Như ta thấy, ở đây vừa có lặp, vừa có rẽ nhánh. Cứ sau mỗi lần lặp, biến AMAX sẽ chứa giá trị lớn nhất của những t_1, t_2,... t_i đã duyệt.

Ta chỉ giữ giá trị lớn nhất chứ không để ý là giá trị lớn nhất đó bằng bao nhiêu. Theo cách làm này, ở lần lặp thứ n (cuối cùng) trong AMAX sẽ chứa giá trị lớn nhất của toàn dãy số.

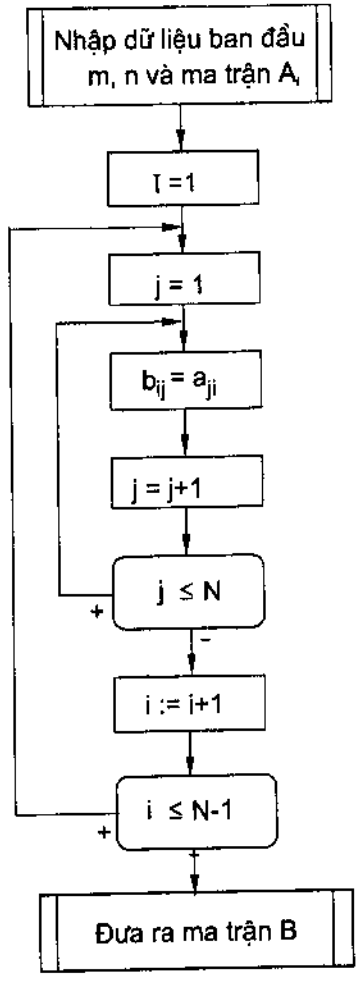


Hình 5.13

Ví dụ 5.12: Cho ma trận vuông:

$$A = \begin{bmatrix} a_{11}a_{12}...a_{n} \\ ..... \\ a_{n1}a_{n2}...a_{nn} \end{bmatrix}$$

Lập sơ đồ thuật toán hoán vị ma trận A. Ma trận kết quả A có các phần tử $b_{ij} = a_{ij}$. Ma trận có hai chiều, các phần tử trên dòng chạy theo chỉ số j, trên cột chạy theo chỉ số i, mỗi cahiều đều biến đổi từ 1 đến n. Hoán vị ma trận thực chất là hoán vị dòng cho cột và ngược lại, phần tử ở dòng i cột j, sẽ trở thành phần tử cột i và dòng j trong ma trận kết quả. Sơ đồ có hai chu trình lồng nhau (hình 5.14).



Hình 5.14

Ví dụ 5.13: Xét ví dụ có ba chu trình lồng nhau: nhân hai ma trận. Cho hai ma trận A, B ban đầu và ma trận C kết quả:

$$A = \begin{bmatrix} a_{11} ...a_{1n} \\ a_{m1}...a_{mn} \end{bmatrix}$$

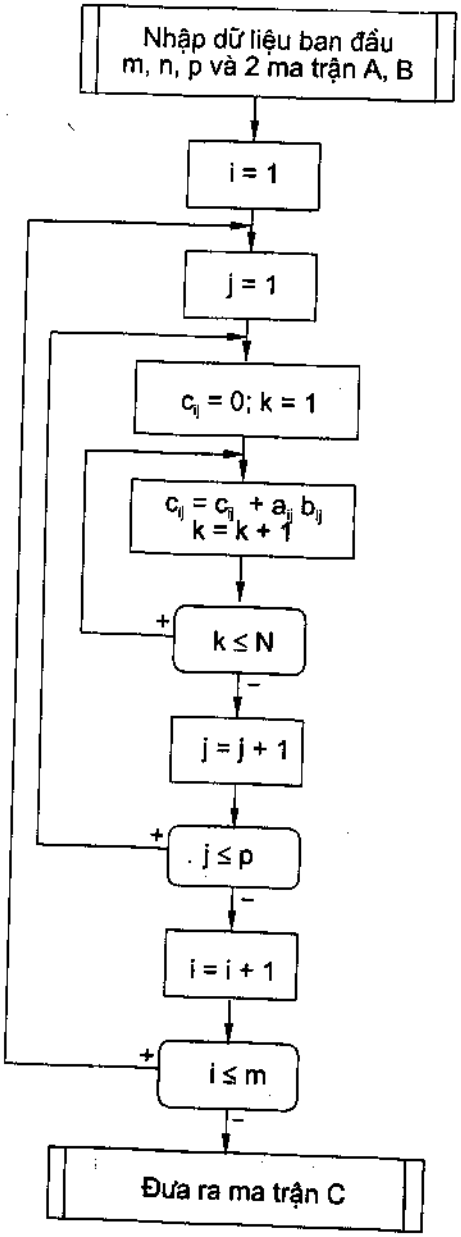
$$B = \begin{bmatrix} b_{11}...b_{1p} \\ a_{n1}...a_{np} \end{bmatrix} \qquad C = \begin{bmatrix} c_{11}...c_{1p} \\ c_{n1}...c_{np} \end{bmatrix}$$

Mỗi phần tử của ma trận C được xác định bằng công thức:

$$C_{ij} = \sum_{k=1}^{n} a_{ik}b_{kj}; \quad i = 1 \div m; \quad j = 1 \div p \qquad (5.5)$$

Trong bài toán này, các phần tử của ma trận C chạy theo hai chỉ số i và j; bản thân mỗi phần tử C_{ij}, lại là một bài toán tính tổng các tích $a_{ik}b_{kj}$

với k chạy từ 1 đến n. Như vậy ta có ba chu trình lồng nhau. Chu trình trong cùng tính tổng C_{ij}.mỗi lần vào chu trình này phải gán lại giá trị ban đầu của nó bằng 0 ($C_{ij} = 0$) và bộ đếm k bắt đầu lại từ (k = 1. Ta có sơ đồ khối như trên hình 5.15.



Hình 5.15

Ví dụ 5.14: Xử lý bảng biểu.

Cho danh sách sinh viên với kết quả học tập như sau (ký hiệu ms là mã số, ht là họ tên, mon i là điểm môn thứ i, $đ_{tb}$ là điểm trung bình, hk là mức hạnh kiểm, $đ_{hk}$ là điểm hạnh kiểm, hb là tiền học bổng)

ms	ht	mon 1		mon p	hk	$đ_{hk}$	$đ_{tb}$	hk

trong đó:

- Cột ms là một mảng, mỗi phần tử là một xâu chín ký tự, gồm hai ký tự mã khoa, hai ký tự mã khoá học, dấu ngăn cách /, hai ký tự mã lớp, hai ký tự mã số thứ tự của sinh viên trong lớp;

- ht là một mảng, mỗi phần tử là một xâu 25 ký tự chữ;

- mon là một mảng, mỗi phần tử là một số thực - điểm môn học, mỗi điểm bốn chữ số, mỗi phần tử là một ký tự chữ (có bốn mức hạnh kiểm A, B, C, D);

- $đ_{tb}$ là một mảng, mỗi phần tử là một số thực - điểm trung bình chung mở rộng, sẽ được tính trong quá trình thực hiện chương trình, theo công thức:

$$đ_{tb} = \frac{\left(\sum_{k=1}^{p} mon_k\right)}{p + đ_{hk}},$$

trong đó: $đ_{hk}$ được tính như sau:

$$đ_{hk} = \begin{cases} 0,6 \text{ nếu là hk là A} \\ 0,3 \text{ nếu là hk là B} \\ 0,6 \text{ nếu là hk là C} \\ -0,3 \text{ nếu là hk là D} \end{cases}$$

- hb là học bổng, được tính như sau: tính suất học bổng trung bình:

shbtb = (quỹ học bổng)/n,

- n là số học sinh được cấp học bổng, bằng tổng số học sinh trừ đi những người bị kỷ luật không được cấp học bổng;

- Tính:

$$y_1 = (\text{số sinh viên có } đ_{tb} < 5,0)/n;$$

$$y_2 = (\text{số sinh viên có } đ_{tb} \text{ từ } 5,0 \text{ đến } 7,0)/n; \quad (*)$$

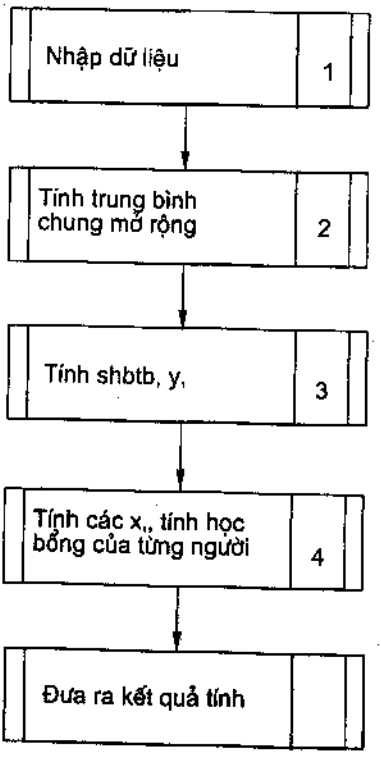
$$y_3 = (\text{số sinh viên có } đ_{tb} > 7,0)/n;$$

Cho suất học bổng trung bình là 100%, ta phải xác định x_1, x_2, x_3, là phần trăm so với shbtb của từng loại y_1, y_2, y_3.

Để cân đối quỹ học bổng, phải xác định điều kiện thoả mãn phương trình:

$$x_1y_1 + x_2y_2 + x_3y_3 = 1 \quad (**)$$

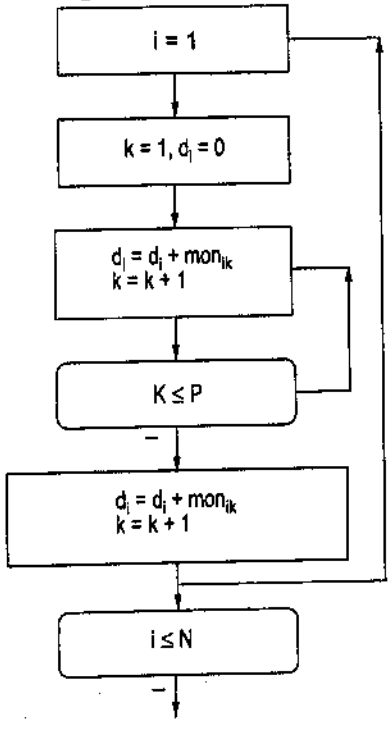
Ba hệ số y_1, y_2, y_3 được xác định từ (*), ta phải cho hai trong ba ẩn x_i giá trị thì mới xác định được ẩn thứ ba, vì chỉ có một phương trình (**). Xuất phát từ nghiệp vụ quản lý sinh viên, ta cho $x_1 = 80\%$; $x_3 = 120\%$. Thay các giá trị này vào (**) ta tìm được x_2. Nhân x_i với shbtb ta xác định được tiền học bổng của từng loại sinh viên. Ta phân tích bài toán ra làm các khối lớn như trên hình 5.16.



Hình 5.16

Để làm ví dụ, ta triển khai một trong các khối trên, khối số 2.

Giả sử trước đó đã xác định điểm hạnh kiểm - $đ_{hk}$, điểm trung bình chung mở rộng ký hiệu là $đ_i$. Hình 5.17 cho ta thuật toán của khối 2. Trong thực tế, việc phân bổ học bổng cho toàn trường rất phức tạp, phải xét nhiều loại đối tượng, nhiều mức điểm khác nhau giữa các lớp, giữa các khoá, trong khuôn khổ chỉ tiêu học bổng hạn chế tuỳ theo từng năm; quỹ học bổng phải chia vừa hết, không được thừa hoặc thiếu.



Hình 5.17

Ví dụ 5.15: Hồi quy tuyến tính.

Phân tích hồi quy là phương pháp thống kê toán học nghiên cứu các mối liên hệ giữa các đại lượng ngẫu nhiên và không ngẫu nhiên, xác định hàm số biểu diễn các mối liên hệ đó.

Trong một quá trình thực nghiệm, ứng với các giá trị x_i ta thu được y_i, $i = 1 \div n$. Khi đường hồi quy thực nghiệm không quá gãy khúc thì ta có thể phán đoán đường cong lý thuyết có dạng tuyến tính $y = ax + b$. Vấn đề đặt ra là phải xác định a và b sao cho:

$$\sum(\bar{y}_i - y_i)^2 \Rightarrow \min,$$

trong đó: $\bar{y}_i$ là giá trị thực nghiệm;

y_i là giá trị tính toán từ phương trình.

$$y_i = ax_i + b.$$

Theo lý thuyết, hàm $\sum(\bar{y}_i - y_i)^2 = \sum(\bar{y}_i - ax_i - b)^2$, đạt cực tiểu khi hàm bậc nhất theo a và b bằng 0:

$$\begin{aligned} \sum(\bar{y}_i - ax_i - b) &= 0 \\ \sum x_i(\bar{y}_i - ax_i - b) &= 0 \end{aligned} \qquad (*)$$

Giải hệ (*).

Từ phương trình thứ nhất ta rút a qua b:

$$a = \frac{\left(\sum \bar{y}_i - nb\right)}{\sum x_i} \qquad (**)$$

Thay a vào phương trình thứ hai, qua biến đổi, ta thu được:

$$b = \frac{\left(\sum x_i^2 - \sum x_i . \bar{y}_i\right)\left(\sum x_i\right)}{\left[n\sum x_i^2 + \left(\sum x_i\right)\left(\sum x_i\right)\right]}$$

Sau khi tìm được b, thay b vào (**) ta tìm được a:

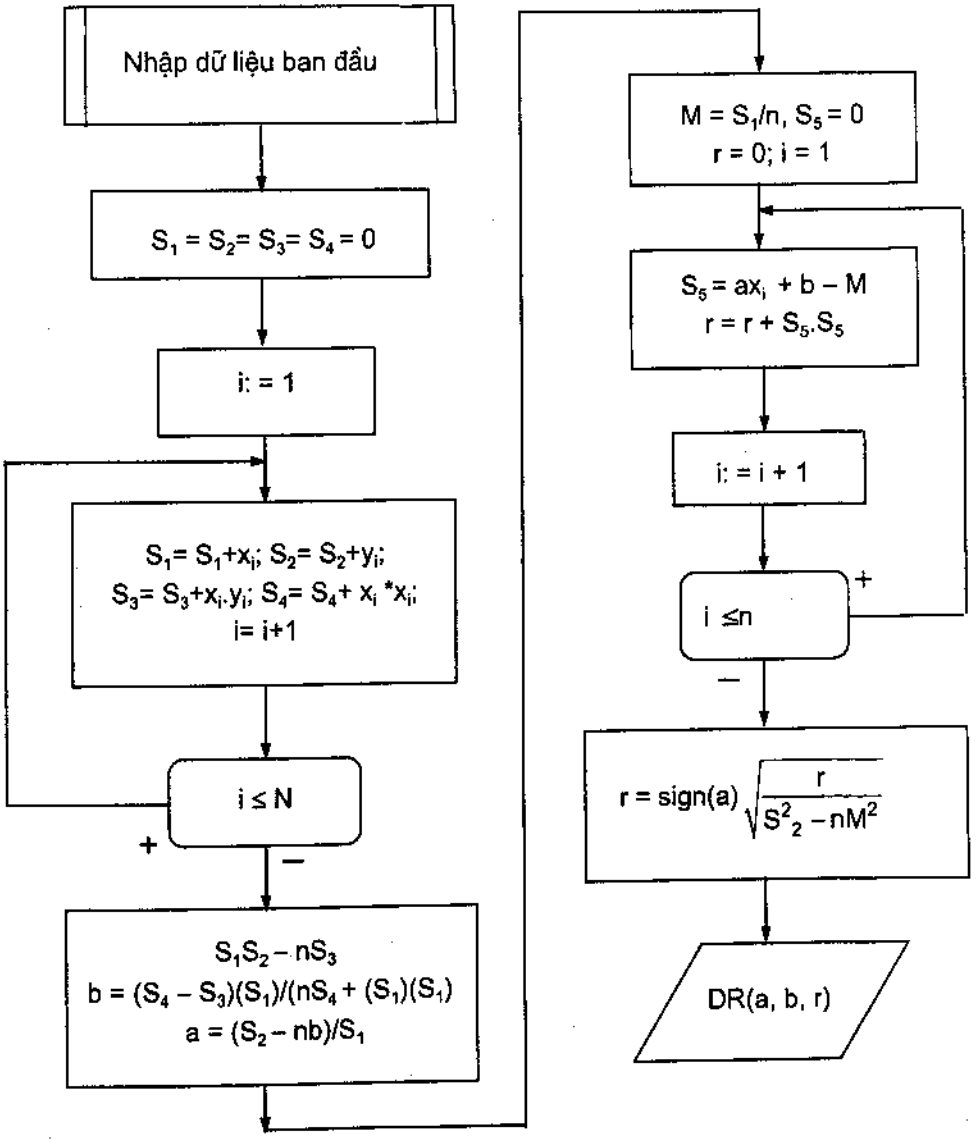
$$a = \frac{\left(\sum \bar{y}_i - nb\right)}{\left(\sum x_i\right)}$$

Sau khi tính được a, b ta cần xác định hệ số tương quan:

$$r = \text{sign}(a)\sqrt{\frac{(y_i - M)^2}{(y_i - M)^2}}$$

Sơ đồ thuật toán như trên hình 5.18.

(M là kỳ vọng toán học).



Hình 5.18

5.5.3. Lặp có số vòng lặp chưa biết trước

Trong thực tế có nhiều bài toán có chu trình không phải là số lần lặp mà là một biểu thức nào đó, chẳng hạn, sai số cho phép.

Ví dụ 5.16: Xét bài toán giải phương trình F(x) = 0 bằng phương pháp lặp. Ta biến đổi phương trình F(x) = 0 về dạng f(x) = x. Với nghiệm ban đầu x_0 ta thiết lập chuỗi.

$x_1 = f(x_0)$,

$x_2 = f(x_1)$,

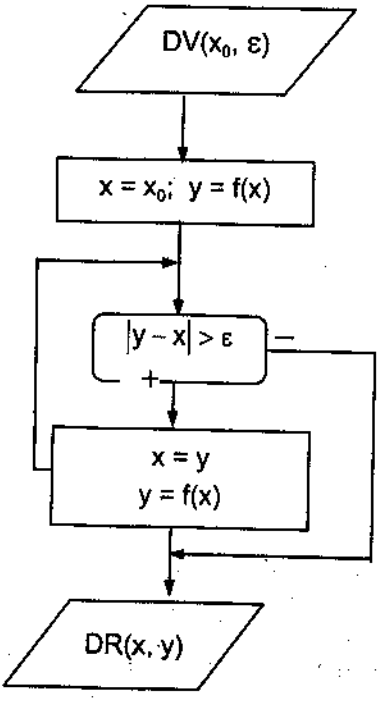
.....

$x_n = f(x_{n-1})$, quá trình dừng khi $|x_n - x_{n-1}| \le \varepsilon$, với ε là giá trị cho trước.

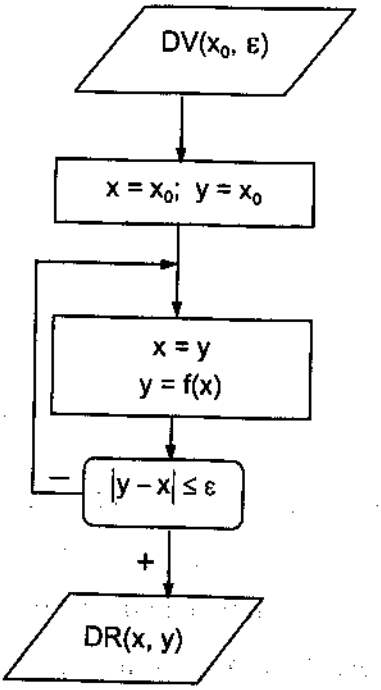
Ở đây ta chỉ mới xét hàm f(x) ở dạng tổng quát.

Số lần tính lặp $x_i = f(x_{i-1})$phụ thuộc vào ε. Vì vậy ta không thể đặt n biến x_i, (n chưa xác định). Khi đó ta đặt một biến x nhận giá trị nghiệm ở mỗi thời điểm tính toán, một biến y nhận giá trị của hàm f; nếu trên mỗi bước lặp chưa thoả mãn biểu thức $|x - y| \le \varepsilon$ thì x sẽ nhận giá trị mới là (x: = y) cho trước lặp tiếp theo.

Hình 5.19 và 5.20 mô tả thuật toán bài toán trên theo hai cách kiểm tra điều kiện trước và sau. Khi tính cụ thể chỉ việc thay công thức của hàm số vào sơ đồ. Bạn đọc hãy làm bài tập với $f(x) = x - \cos x + 0{,}7$; $x_0 = 1$, $\varepsilon = 10^{-6}$.



Hình 5.19



Hình 5.20

Ví dụ 5.17: Giải hệ phương trình đại số theo phương pháp Gauss.

Cho hệ phương trình đại số tuyến tính:

$$\begin{aligned} &a_{1,1}x_1 + a_{1,2}x_2 + \ldots + a_{1,n}x_n = a_{1,n+1} \\ &a_{2,1}x_1 + a_{2,2}x_2 + \ldots + a_{2,n}x_n = a_{2,n+1} \\ &\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ &a_{n,1}x_1 + a_{n,2}x_2 + \ldots + a_{n,n}x_n = a_{n,n+1} \end{aligned} \qquad (*)$$

Theo phương pháp Gauss, từ hệ thức (*) ta đưa về dạng tam giác:

$$\begin{aligned} a_{1,1}x_1 + a_{1,2}x_2 + \ldots + a_{1,n}x_n &= a_{1,n+1} \\ a_{2,2}x_2 + \ldots + a_{2,n}x_n &= a_{2,n+1} \\ \ldots &= \ldots \\ a_{n,n}x_n &= a_{n,n+1} \end{aligned} \qquad (**)$$

Từ (**) ta sẽ tính ngược được vectơ nghiệm:

$$x_n = a_{n,n+1}/a_{n,n}$$

$$x_{n-1} = (a_{n-1,n-1} - a_{n-1,n} * x_n)/a_{n-1,n-1} \ldots$$

Như vậy thuật toán có hai giai đoạn: đưa ma trận ban đầu về dạng tam giác; giải ngược tìm nghiệm.

a) Giai đoạn thứ nhất:

1. Bắt đầu từ j = 1

2. Nếu $a_{j,j} \neq 0$ thì dùng cách tổ hợp tuyến tính sẽ khử các phần tử ở cột j kể từ hàng thứ j + 1 đến n, sau đó tăng j = j + 1. Nếu $j \leq n - 1$ thì trở lại mục 2, nếu $j > n - 1$ thì trở xuống mục 4.

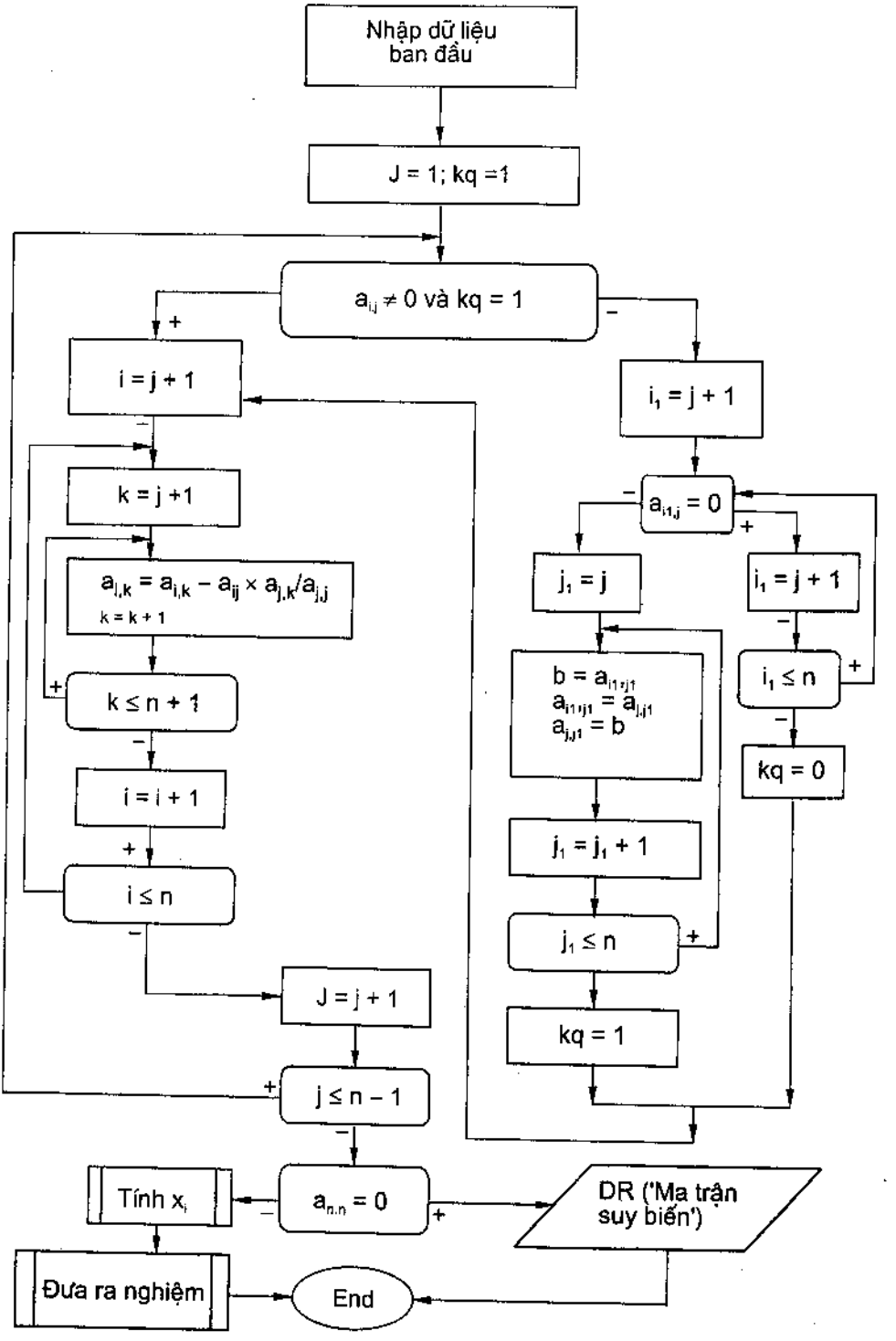
3. Nếu $a_{j,j} = 0$ phải tìm trong cột j một phần tử $a_{k,j} \neq 0$ và đổi chỗ hai hàng k, j cho nhau;

4. Nếu $a_{n,n} = 0$ thì suy ra ma trận A suy biến, hệ không có nghiệm duy nhất, dừng chương trình, nếu $a_{n,n} \neq 0$, xong giai đoạn thứ nhất.

b) Giai đoạn thứ hai:

Tính các $x_n, x_{n-1}, \ldots$

Sơ đồ khối như trên hình 5.21.


Nhập dữ liệu ban đầu
J = 1; kq =1
$a_{i,j} \neq 0$ và kq = 1
+
−
i = j + 1
k = j +1
$a_{l,k} = a_{i,k} - a_{ij} \times a_{j,k}/a_{j,j}$
k = k + 1
k ≤ n + 1
i = i + 1
i ≤ n
J = j + 1
j ≤ n − 1
Tính x_i
$a_{n,n} = 0$
DR ('Ma trận suy biến')
Đưa ra nghiệm
End
$i_1 = j + 1$
$a_{i1,j} = 0$
$j_1 = j$
$i_1 = j + 1$
b = $a_{i1,j1}$
$a_{i1,j1} = a_{j,j1}$
$a_{j,j1} = b$
$j_1 = j_1 + 1$
$j_1 \leq n$
kq = 1
$i_1 \leq n$
kq = 0

Hình 5.21

Chương 6

PHÂN TÍCH THUẬT TOÁN

6.1. KHÁI NIỆM

Mỗi thuật toán phải đảm bảo các tính chất:

- Các bước phải được xác định rõ ràng, rành mạch từ bắt đầu đến kết thúc;
- Tổng quát, xét được mọi khả năng, mọi trường hợp của bài toán;
- Chi tiết, trong từng khả năng, từng trường hợp phải xác định được công thức tính toán hay dòng thông báo về kết quả xử lý;
- Dễ cài đặt chương trình máy tính;
- Hữu hạn: thời gian thực hiện thuật toán chấp nhận được.

Trong chương trước, ta đã học các phương pháp trình bày thuật toán, các dạng thuật toán cơ bản. Một bài toán có thể có nhiều cách giải khác nhau, có thể có nhiều thuật toán khác nhau, việc chọn thuật toán nào để viết chương trình cho máy tính đã được các chuyên gia tin học đề cập đến từ lâu, và một lĩnh vực mới đã hình thành: nghiên cứu và phân tích độ phức tạp, độ hiệu quả của thuật toán.

Ví dụ: Xét một thuật toán điển hình:

Bài toán tìm kiếm nhị phân.

Cần xác định phần tử T có trong mảng được sắp X[1..N] hay không.

0. Đưa vào T
1. Found =.false.
2. Đầu = 1
3. Cuối = N
4. WHILE đầu ≤cuối AND not found DO

{

a) M = (đầu + cuối)/2

```
b) IF T<X[M] THEN
                    cuối : = M-1
               ELSE
               { IF   T>X[M] THEN
                                   Đầu M+1
                              ELSE
                              Found =.true.
               }
}
```

Ở đây X[1..N] có thể là số, có thể là xâu ký tự chữ (xem mảng - array).

Những vấn đề cần quan tâm khi xây dựng thuật toán:

+ Thuật toán phải đảm bảo được 5 tính chất như đã nêu ở đầu mục 6.1 này;

+ Cách trình bày thuật toán phải giúp cho việc viết đúng chương trình;

+ Cách trình bày thuật toán phải giúp cho việc lựa chọn kỹ thuật lập trình để nâng cao hiệu suất của chương trình.

6.2. ĐỘ HIỆU QUẢ VÀ ĐỘ PHỨC TẠP CỦA THUẬT TOÁN

6.2.1. Các loại bài toán

Độ phức tạp của thuật toán thường phụ thuộc vào từng loại bài toán cần giải quyết. Có bài toán có khối lượng xử lý thông tin rất lớn (như bài toán thống kê dân số, ...), có bài toán có ít thông tin ban đầu nhưng lại rất phức tạp về mặt toán học, logic tính toán. Ta gọi chung các bài toán cần xử lý trên máy tính là công việc tính toán.

Ta có thể phân chia các công việc tính toán cần xử lý trên máy tính ra các lĩnh vực sau:

- Quản lý doanh nghiệp, quản lý nhà nước - chính phủ và quản lý ở các cấp bộ ngành thuộc chính phủ, quản lý nhà nước ở cấp tỉnh và các sở thuộc tỉnh, quản lý tài chính - kế toán - ngân hàng, quản lý tư pháp, quản lý tòa án, quản lý giáo dục đào tạo.

- Khoa học kỹ thuật thuần tuý.

- Lĩnh vực khoa học cơ bản.

- Trong cơ khí, kiến trúc, xây dựng nhà, xây dựng cầu, đường, thuỷ lợi, giao thông, nông nghiệp v.v...

- Thuần tuý tin học: dịch vụ tra cứu thông tin trên mạng, thư tín điện tử, thương mại điện tử v.v...

- Lĩnh vực trò chơi, dạy học, du lịch, dịch vụ công cộng v.v...

6.2.2. Xác định độ hiệu quả, độ phức tạp của thuật toán

Các cấu trúc dữ liệu có thể được cài đặt trong bộ nhớ máy tính theo nhiều cách khác nhau, mỗi cách đòi hỏi phải dùng số lượng bộ nhớ và thời gian tính toán nhất định. Cách nào mà sử dụng ít bộ nhớ nhất và thời gian tính toán ít nhất thì nó là hiệu quả nhất. Người ta chỉ quan tâm đến thời gian tính toán. Yếu tố quan trọng nhất ảnh hưởng đến thời gian tính toán là khối lượng thông tin đầu vào và kỹ thuật lập trình được sử dụng.

Ký hiệu t là thời gian thực hiện của thuật toán, khi đó t là một hàm số phụ thuộc vào số lượng đầu vào n, nên ta viết là T(n).

Khái niệm độ phức tạp của thuật toán được hiểu theo ý nghĩa là độ lớn của T(n), được ký hiệu bằng chữ O (đọc là ô lớn) như sau: giả sử n là số nguyên dương, T(n) và f(n) là các hàm thực không âm, khi đó $T(n) = O(f(n))$ *nếu và chỉ nếu tồn tại các hằng dương c và* n_0 *sao cho* $T(n) \leq cf(n)$ *với mọi* $n \geq n_0$.

Theo định nghĩa trên, hàm f(n) là cận trên của T(n). Thuật toán có thời gian thực hiện T(n) = O(f(n))- ta nói rằng nó có thời gian thực hiện cấp f(n).

Trong lý thuyết phân tích thuật toán, người ta đã đưa ra một số cấp thời gian chuẩn thực hiện thuật toán như sau:

O(1) - thời gian thực hiện bị chặn bởi 1 hằng số;

O(logn) - thời gian thực hiện theo logarit

O(nlogn) - thời gian thực hiện theo nlogn

O(n) - thời gian thực hiện tuyến tính

$O(n^2)$ - thời gian thực hiện bình phương

$O(2^n)$ - thời gian thực hiện là số mũ

Với các chỉ tiêu trên đây, ta đã có một độ đo để đánh giá, so sánh thuật toán nào tốt hơn thuật toán nào khi giải quyết một bài toán.

6.2.3. Quy tắc đánh giá thời gian thực hiện thuật toán

Quy tắc tổng: nếu $T_1(n) = O(f_1(n))$ và $T_2(n) = O(f_2(n))$

thì $T_1(n) + T_2(n) = O(\max(f_1(n), f_2(n))$

Thật vậy:

vì $T_1(n) = O(f_1(n))$ nên $\exists\ c_1, n_1$ sao cho $T_1(n) \le c_1 f_1(n))$ với $\forall\ n \ge n_1$,

$T_2(n) = O(f_2(n))$ nên $\exists\ c_2, n_2$ sao cho $T_2(n) \le c_2 f_2(n))$ với $\forall\ n \ge n_2$,

Đặt $n_0 = \max(n_1, n_2)$, khi đó với $\forall\ n \ge n_0$ ta có:

$$T_1(n) + T_2(n) \le (c_1 + c_2)\max(f_1(n), f_2(n))$$

Có thể nói một cách tổng quát như sau: khi một bài toán được chia thành nhiều bài toán con thì độ đo cấp độ thời gian thực hiện thuật toán sẽ lấy theo cấp độ lớn nhất trong các cấp độ thành phần.

6.2.4. Đánh giá thời gian thực hiện các câu lệnh Pascal

Để đánh giá thời gian thực hiện một chương trình, ta phải biết được từng câu lệnh thực hiện bao nhiêu lần.

Tuỳ theo loại lệnh, ta có các chỉ tiêu đánh giá như sau:

1. Lệnh gán, lệnh read, write, goto có thời gian thực hiện là O(1);

2. Lệnh phức hợp để trong cặp begin... end được đánh giá theo quy tắc tổng ở trên;

3. Lệnh IF <điều kiện> THEN S1 ELSE S2, thời gian thực hiện là:

$$O(\max(f_1(n), f_2(n))$$

trong đó $f_1(n)$, $f_2(n)$ là cận trên của thời gian thực hiện lệnh S1, S2.

4. Lệnh CASE được đánh giá như lệnh IF.

5. Lệnh lặp (repeat...until, while..do. for...do): nếu thời gian thực hiện thân vòng lặp là f(n) và số lần lặp tối đa là g(n) thì thời gian thực hiện một lệnh lặp là O(f(n)g(n)).

6. Đối với hàm đệ quy:

Xét hàm đệ quy tính giai thừa sau:

```
Function giaithua(n:integer):integer;
Begin if n < = 1 then giaithua : = 1
                 else giaithua : = n*giaithua(n-1);
End;
```

Ta đặt thời gian thực hiện hàm với n là T(n). Với n = 1 thì thời gian thực hiện hàm là O(1). Với n >1, thực hiện hàm chính là thời gian thực hiện lệnh gán giaithua: = n*giaithua(n – 1), tức là bằng O(1) cộng với T(n–1). Ta có quan hệ đệ quy:

T(1) = O(1);

T(n) = O(1) + T(n – 1)

Ta bắt đầu từ T(n) cho đến T(n – 1), T(n – 2),... T(2), T(1):

$T(n) = O(1) + O(1 + ... + O(1) + T(1) = (n - 1)O(1) + T(1) = nc_1 = O(n)$.

6.2.5. Phân tích một số thuật toán

6.2.5.1. Hàm fibonacy

```
Function fibonacy(n:integer):integer;
Begin If n<2 then fibonacy : = n
                 else fibonacy : = fibonacy(n-1) + fibonacy(n-2);
End;
```

```
                    1
              1           2
        1           2           3
    1       2       3       5
1       2       3       5       8
.....
```

Ta có quan hệ:

Với n < 2 và 1 hằng số c, T(n) = c

Với n ≥2, T(n) = T(n – 1) + T(n – 2)

Người ta đã tính toán thời gian thực hiện:

$$T(n) = O(\phi^n), \text{ với } \phi = \frac{(1+\sqrt{5})}{2}$$

Thời gian thực hiện này là hàm mũ, vì vậy với n đủ lớn thì hàm này không hiệu quả.

6.2.5.2. Thuật toán Euclid - tìm ước số chung

```
Function Euclid(m, n:integer):integer;
Var r:integer;
Begin
    r: = m mod n;
    While r <> 0 do
                Begin m: = n;
                      n : = r;
                      r : = m mod n;
                end;
    Euclid : = n;
End;
```

Xét thời gian thực hiện trong thân thuật toán:

$$m = n.q_1 + r_1,\ 0 \le r_1 < n$$

$$n = r_1.q_2 + r_2,\ 0 \le r_2 < r_1$$

Nếu $r_1 > n/2$ thì suy ra $q_2 = 1$, tức là $n = r_1 + r_2 \Rightarrow r_2 < n/2$

Còn $r_1 < n/2$ thì cũng suy ra $r_2 < n/2$.

Nghĩa là trong mọi trường hợp, $r_2 < n/2$.

Như vậy, cứ sau hai lần thực hiện thân vòng lặp thì số dư sẽ giảm đi một nửa của n.

Giả sử k là số nguyên lớn nhất sao cho $2^k \le n$, số lần lặp tối đa sẽ là 2k+1.

Từ bất đẳng thức $2^k \le n$ ta có:

$$k \le \log_2 n \Rightarrow 2k \le \log_2 n \Rightarrow 2k+1 \le \log_2 n + 1$$

Thời gian thực hiện thuật toán sẽ là $O(\log_2 n)$.

Chương 7

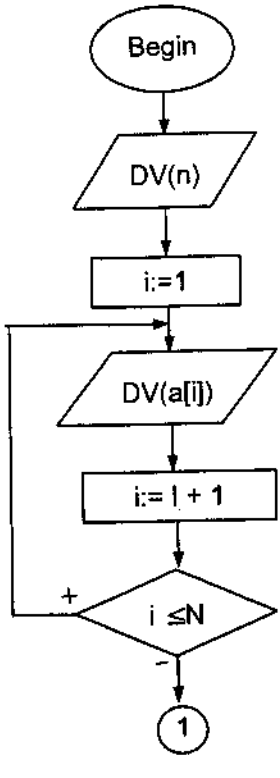
CÁC THUẬT TOÁN SẮP XẾP

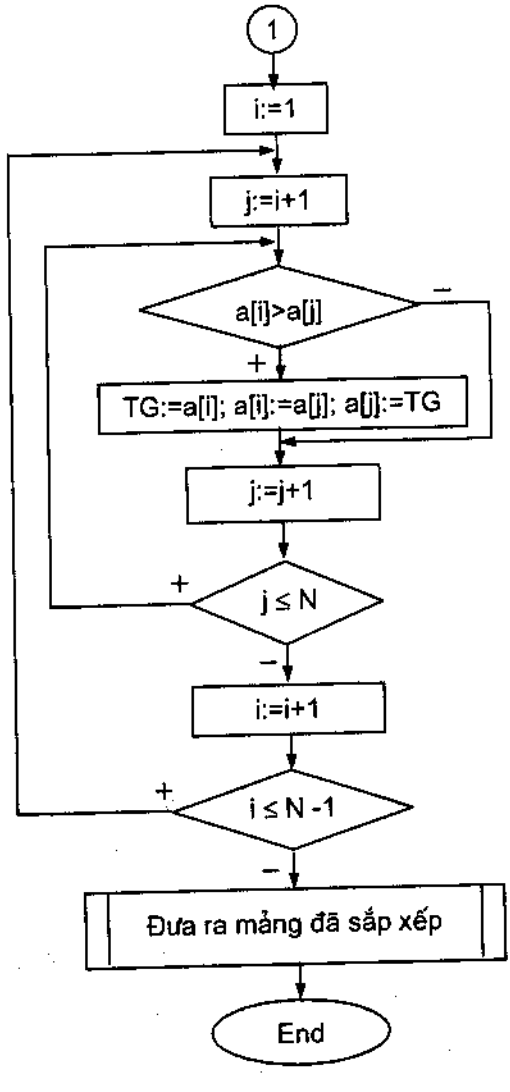
7.1. SẮP XẾP BẰNG CHỌN LỰA ĐƠN GIẢN CHO MẢNG

7.1.1. Bài toán

Cho mảng x_1, ..., x_n các số thực bất kỳ, lập thuật toán sắp xếp lại mảng theo thứ tự tăng dần.

7.1.2. Sơ đồ thuật toán





Đoạn cốt lõi của chương trình như sau:

```
for  I := 1 to n-1 do
  for  j :=  i+1 to n do
     if x[i] <x[i] then
          begin x[i] := x[i] +x[j]
                x[j]:=x[i]-x[j]
               x[i]:= x[i] -x[j]
          end;
```

7.2. SẮP XẾP THEO CHỌN LỰA ĐƠN GIẢN CHO DANH SÁCH LIÊN KẾT

7.2.1. Bài toán

Cho một danh sách liên kết. Yêu cầu lập thuật toán sắp xếp các phần tử trong danh sách liên kết đó theo thứ tự tăng dần.

7.2.2. Thuật toán

Ở đây ta sử dụng con trỏ P để xử lý và con trỏ SmallPtr để lưu phần tử nhỏ nhất, mỗi con trỏ là một bản ghi có 2 trường: Du_lieu và Next.

1. Khởi động con trỏ P tại nút đầu tiên

2. While P < > Nil làm các bước sau:

a) Đặt con trỏ SmallPtr : = P

b) Đặt Smallest : = Du_Lieu(SmallPtr);

c) Đặt con trỏ q : = Next(p)

d) While q < > nil then begin

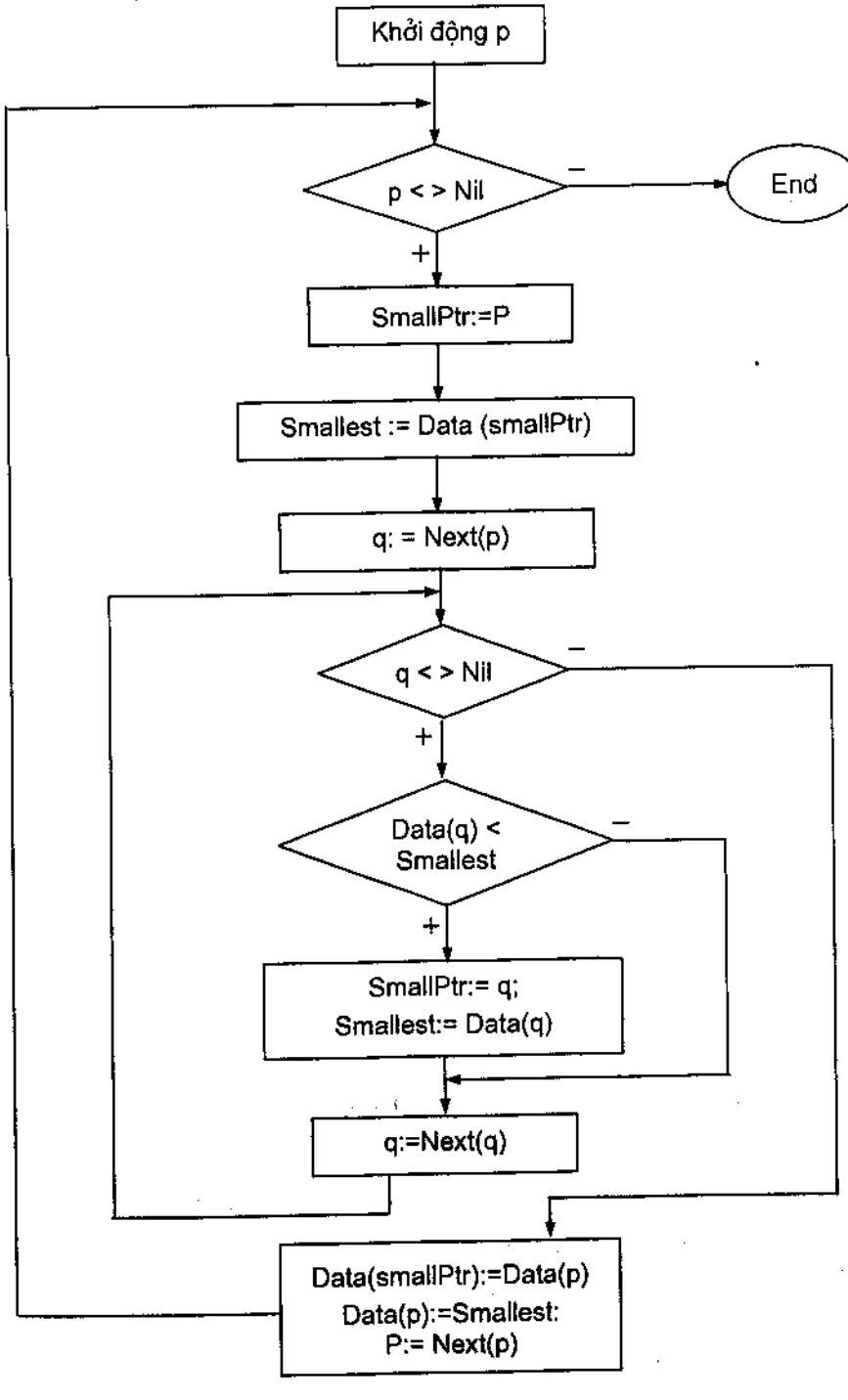
```
                    if Du_Lieu(q) < Smallest then
    begin SmallPtr : = q; Smallest : = Du_Lieu(q) end;
                    q : = Next(q)
                                end
```

e) Du_Lieu(SmallPtr) : = Du_Lieu(p)

f) Du_Lieu(p) : = Smallest

g) p : = Next(p)

Sơ đồ thuật toán:



Theo sơ đồ của Lany NuHoff (Tài liệu tham khảo số 2).

```
For  i: = 1 to n-1 Do
    begin  smallpos :=  i;
           Smallest :=  X[smallpos];
           for j : = i+1 to  n  Do
             if  x[j] < smallet then
                begin Smallpos := j;
                      smallest := X[smallpos]
                end;
             X[smallpos]:=X[i];
             x[i]:= smallest;
     end;
```

Độ phức tạp $(n-1)+(n-2)+...+2+1=\frac{n(n-1)}{2}=O(n^2)$

7.3. SẮP XẾP KIỂU NỔI BỌT - SẮP XẾP DẦN

7.3.1. Xét bài toán

Cho dãy các số thực x_1, x_2, x_3, ..., x_n, yêu cầu sắp xếp lại theo trật tự tăng dần theo kiểu nổi bọt.

7.3.2. Phân tích bài toán

Sắp xếp nổi bọt được thực hiện theo thuật toán: trong mỗi lần đi qua nếu không có thứ tự đúng thì sắp xếp ngay hai phần tử liền kề. Theo phương pháp này, mỗi lần duyệt qua một lượt danh sách, một số phần tử lớn được nổi lên trên, một số phần tử bé bị chìm xuống dưới.

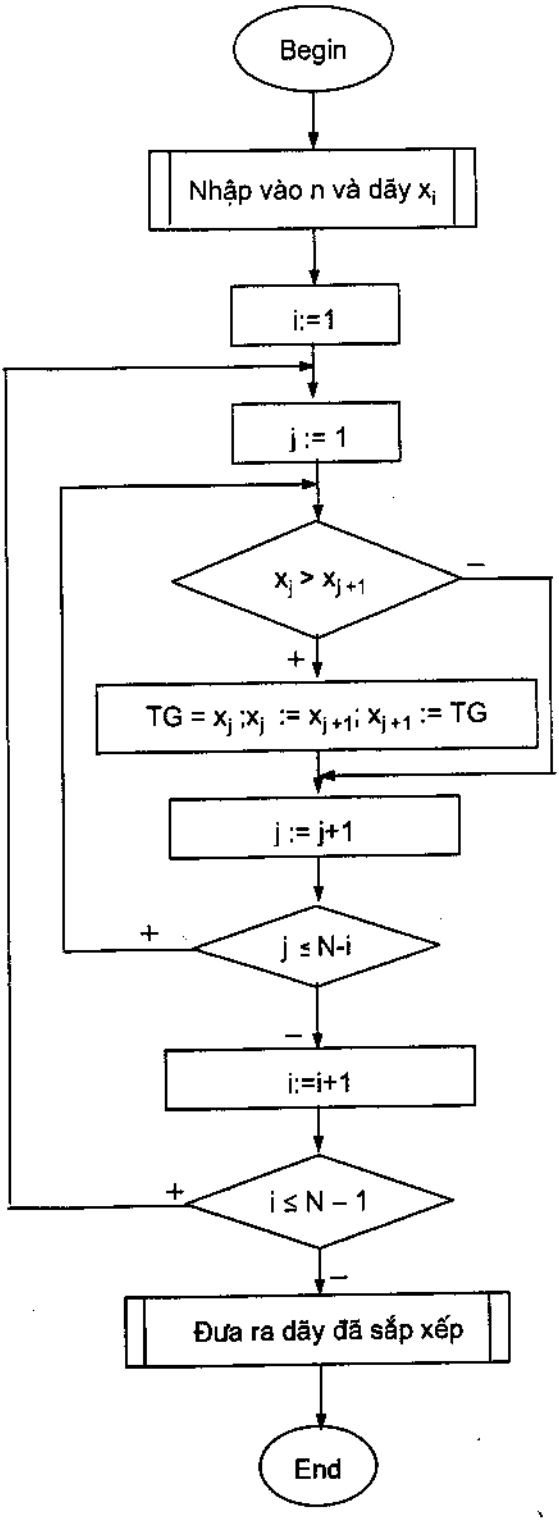
Bắt đầu từ so sánh x_1 và x_2. Nếu $x_1 > x_2$ thì hoán vị luôn x_1 và x_2:

x_2, x_1, x_3..., x_n $\Rightarrow$ hoán đổi chỉ số, x_1 ở vào vị trí x_2 và ngược lại:

kết quả: $\Rightarrow$ x_1, x_2, x_3..., x_n

Tiếp tục so sánh x_2 với x_3... cho đến x_{n-1} và x_n

Như vậy kết thúc lần đi qua thứ nhất, phần tử lớn nhất đã "nổi" lên cuối danh sách, và một vài phần tử đã "chìm" về phía gần đầu danh sách.


Begin
Nhập vào n và dãy x_i
i:=1
j := 1
$x_j > x_{j+1}$
+
−
TG = x_j ;x_j := x_{j+1}; x_{j+1} := TG
j := j+1
+
j ≤ N-i
−
i:=i+1
+
i ≤ N – 1
−
Đưa ra dãy đã sắp xếp
End

Tiếp tục quá trình này, các phân tử lớn nhất trong danh sách con còn lại sẽ ở vào cuối cùng của danh sách con đó.

Ta có thuật toán sắp xếp theo trật tự tăng dần như ở trang trên. Chương trình sau đây viết và chạy trên Pascal for Windows:

```
Program sapxep_noibot;
uses WinCrt;
var i,j,n,k:integer;
   a:array[1..50] of real;
   tg:real;
Begin write("Nhap N:");readln(N);
    For i:= 1 to N Do
       Begin write("Nhap a[",i,"]= ");readln(a[i]) end ;
    For i:= 1 to N do
    For j:=1 to n-i do if a[j]> a[j+1] then
                    begin tg:=a[j];
                        a[j]:=a[j+1];
                        a[j+1]:=tg
                    end;
    For i:= 1 to N do  writeln(a[i]:8:2," ");
End.
```

7.4. SẮP XẾP KIỂU CHÈN

7.4.1. Bài toán

Thuật toán này dùng trong trường hợp khi ta tạo một danh sách, ví dụ, sắp xếp một danh sách chứa các phân tử trong mảng x_1, x_2, ..., x_n theo thứ tự tăng dần. Các phần tử đưa vào danh sách được đặt đúng vị trí theo trật tự giá trị tăng dần. Chú ý là các x_1, x_2, ..., x_i được đưa vào cho đến thời điểm thứ i.

7.4.2. Phân tích

Ví dụ: ta có các số 67, 33, 21, 84, 49, 50, 75, ta sẽ đưa vào danh sách và sắp xếp theo chiều tăng dần:

Bước 1: đưa vào số 67

Bước 2: đưa vào số 33: so sánh và chèn phía trước: 33, 67

Bước 3: đưa vào 21: chèn vào trước: 21, 33, 67

Bước 4: đưa vào 84: chèn phía sau: 21, 33, 67, 84

Bước 5: i = 5, j = i-1 = 4. đưa vào số 49: kiểm tra từ cuối danh sách đã đưa vào, vì 49 < 84 nên số 84 được dịch chuyển sang phải 1 vị trí thứ 5 để trống vị trí thứ 4. Giảm j để xét phần tử thứ 3, vì 49 < 67 nên dịch chuyển 67 lên vị trí thứ 4 để trống vị trí thứ 3. Giảm j, xét phần tử thứ 2, vì 33 < 49 nên không dịch chuyển nữa mà ra khỏi lệnh while và gán số 49 vào vị trí thứ 3.

Các bước tiếp theo: quá trình lặp cho đến hết các phần tử cần đưa vào.

Giải thích sơ đồ thuật toán:

1. Đầu tiên cần nhập vào số lượng phần tử N

2. Bắt đầu nhập phần tử thứ nhất x[1]

3. Cho I chạy từ 2 (i:= 2), nhập số mới

4. So sánh: nếu số mới nhập ≥x[1] thì x[i]: = <số mới>

nếu không thì x[i]:= x[i-1] và x[i-1]: = <số mới>

5. Tăng i: = i+1

6. Nhập số mới

j: = i – 1

Nếu số mới ≥x[j] thì:

{x[j + 1] := <số mới>; goto (5) }

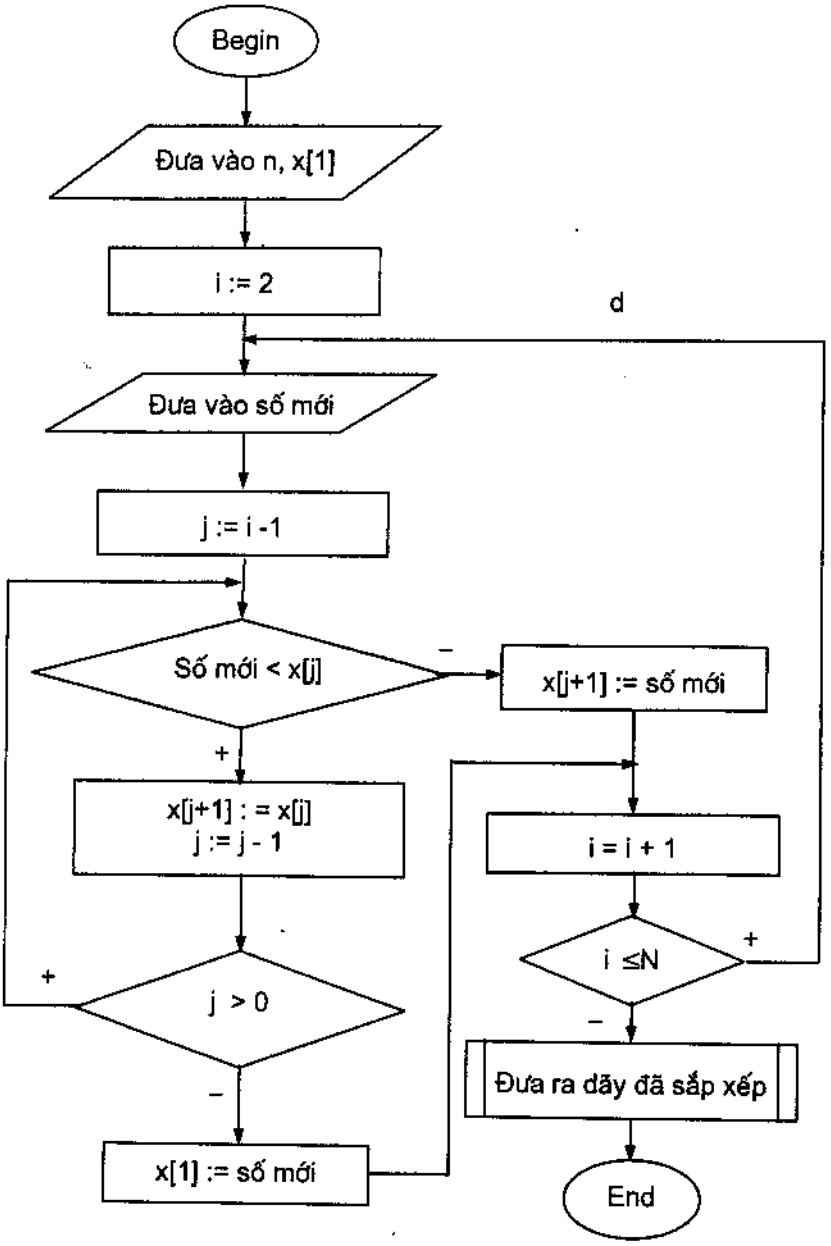
Nếu không thì:

{x[j + 1] := x[j + 1]; j : = j – 1 }

Kiểm tra: Nếu j > 0 thì goto (5), nếu không thì x[1]: = <số mới>

7. Kiểm tra nếu i ≤ n thì goto (4) để tăng i tiếp tục, nếu không thì đưa ra dãy đã sắp xếp và kết thúc.

Ta có sơ đồ thuật toán:



7.5. SẮP XẾP NHANH

7.5.1. Giải thích phương pháp

Thuật toán sắp xếp nhanh một dãy số do C.A.R Hoare đề xướng. được giải thích như sau:

- Bước 1: chọn 1 phần tử của danh sách làm mốc.

- Bước 2: phân hoạch dãy thành 2 phần, phần đầu gồm những phần tử có giá trị nhỏ hơn hoặc bằng phần tử mốc, phần cuối gồm những phần tử lớn hơn giá trị phần tử mốc. Kết thúc một lần phân hoạch, phần tử mốc được đặt đúng vị trí thực sự của nó và danh sách được chia làm 2 phần.

Trong quá trình phân hoạch, tại mỗi thời điểm chỉ thực hiện ở một phân đoạn, còn một phân đoạn khác phải được ghi nhớ lại. Quá trình xử lí một phân đoạn, ghi nhớ một phân đoạn được tiếp tục cho tới khi trong một phân đoạn chỉ còn 1 phần tử. Sau đó thực hiện tiếp cho phân đoạn khác. Quá trình dừng khi phân đoạn cuối cùng được xử lí.

***Ví dụ*:** Cho dãy 10 số

a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}
40	19	68	15	64	57	91	33	98	89

Đầu tiên ta chọn mốc là $a_1 = 40$.

Bước phân hoạch thứ nhất:

Cho l = 1, k = 11. Tăng l : = l+1 cho tới khi a[l] > 40.

Giảm k: = k-1 cho tới khi a[k] ≤40.

Đi từ bên trái, trong các phần tử trên: $a_2 = 19 < 40$, chỉ với l = 3, $a_3 = 68 > 40$.

Đi từ bên phải, $a_{10} > 40$, $a_9 > 40$, chỉ với k = 8, $a_8 < 40$. Vì $l = 3 < k = 8$ nên đổi chỗ a_3 cho a_8:

a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}
40	19	33	15	64	57	91	68	98	89

Tiếp tục tăng l và giảm k:

$l = 4$, $a_4 < 40$; $l = 5$, $a_5 = 64 > 40$.

Từ bên phải: $a_7 = 91 > 40$, $a_6 = 57 > 40$, với k = 4, $a_4 = 15 < 40$. Nhưng vì $l = 5 > k = 4$ nên phải đặt lại mốc, hoán vị a_1 với a_4:

a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}
15	19	33	40	64	57	91	68	98	89

Kết quả của bước thứ nhất cho ta 2 phân đoạn:

15	19	33	40

và

64	57	91	68	98	89

Ta có thủ tục phân hoạch như sau:

```
Procedure PH(i, j:integer ; var k:integer);
Var   L, p : integer;
Begin
      p:= a[i];
      L:= i; k:= j+1;
      Repeat L:= L+1 Until (a[L] > p) or (L>j);
      Repeat k:= k-1 Until a[k] < = p;
      While L < k do
          Begin doicho(a[L], a[k]);
              Repeat L:= L+1 Until (a[L] > p) or (L> j) ;
                        Repeat k: = k - 1 Until a[k] < = p
          end;
      If L>k Then doicho(a[i], a[k]);
End;
```

7.5.2. Chương trình

```
Program SapXepNhanh;
Type mang = array[1..50] of Integer;
var
      m, n :integer;
      a:mang;
Procedure Doicho(var w, z:integer);
var tg:integer;
Begin tg: = w; w: = z; z: = tg End;
```

```
Procedure PH(i, j:integer; var k:integer);
Var   L, p : integer;
Begin
    p: = a[i];L:= i; k:= j+1;
    Repeat L:= L+1 Until a[L] > p or (L>j);
    Repeat k:= k-1 Until a[k] ≤p;
    While L < k do
            Begin doicho(a[L], a[k]);
            Repeat L:= L+1 Until (a[L] > p) or (L>j) ;
            Repeat k:= k - 1 Until a[k] ≤p
            end;
    If L>k Then doicho(a[i], a[k]);
End;
Procedure QS(i, j:Integer);
Var k:Integer;
Begin If i < j Then
Begin PH(i, j, k);QS(i, k-1); QS(k+1, j) End;
End;
Begin {Chuong trinh chinh}
    Write("Nhap N:"); readln(N);
    For m:= 1 To N Do
      Begin Write("Nhap a[", m, "]=");Readln(a[m]) end; QS(1, n);
For m: = 1 to N-1 Do write(a[m]:3, ", ");writeln(a[n]:3);readln
End.
```

7.8. SẮP XẾP TRỘN - MERGESORT

7.8.1. Giới thiệu phương pháp

Ý tưởng của phương pháp sắp xếp hòa trộn cũng gần giống với phương pháp sắp xếp nhanh và phương pháp tìm kiếm nhị phân: để sắp xếp các phần tử của dãy a[p, q] thì ta xác định vị trí k = (p + q) div 2 nằm giữa p, q, sắp xếp 2 dãy con a[p, k] và a[k, q] sau đó hòa trộn 2 dãy con này thành dãy được sắp xếp.

7.8.2. Ghép 2 dãy đã sắp thứ tự thành 1 dãy sắp thứ tự

Phần cốt lõi của sắp xếp theo kiểu trộn là thuật toán trộn 2 dãy đã sắp thứ tự thành 1 dãy được sắp.

Ví dụ, ta có dãy: a = 1, 3, 5, 7, 8

và dãy b = 2, 4, 6, 9, 10

Kết quả sau khi trộn: c = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10.

Có nhiều sơ đồ ghép khác nhau, sau đây ta nêu một trường hợp:

```
Program SXTron;
{Tron 2 day da sap thu tu thanh 1 day sap thu tu}
uses dos;
var     i,j,k,n,p,L :integer;
    a,b:array[1..20] of integer;
    c:array[1..40] of integer;
begin
    write("nhap n=");readln(n);
    k:= (1+n) div 2;writeln("k=",k:3);
    for i:=1 to k do
      begin
          write("nhap a[",i,"]=");readln(a[i])
      end;
    for i:=1 to (n-k) do
      begin
          write("nhap b[",i,"]=");readln(b[i])
      end;
   p:=0;i:=1;j:=1;
  while (i<= k) and (j<=(n-k)) do
      begin
          p:=p+1;
            if a[i]<=b[j] then
                begin c[p]:=a[i];i:=i+1; end
                    else
```

```
                    begin c[p]:=b[j];j:=j+1 end;
        end;
        i:=i-1;j:=j-1;
    if i<k then
        for L:=i+1 to k do
            begin p:=p+1;c[p]:=a[L] end;
    if j<(n-k) then
        for L:=j+1 to (n-k) do
            begin p:=p+1;c[p]:=b[L] end;
    writeln(" Day sau tron : ");
        for i:=1 to n-1 do write(c[i]:3," ,");
    write(c[n]:3);
  readln
end.
```

Chương 8
CÁC THUẬT TOÁN TÌM KIẾM

8.1. TÌM KIẾM TRONG DANH SÁCH

8.1.1. Tìm phần tử trong một danh sách tuyến tính chưa được sắp thứ tự

8.1.1.1. Bài toán

Cho một danh sách có n phần tử chưa được sắp xếp và một giá trị T, yêu cầu tìm xem có T trong danh sách đó không.

8.1.1.2. Thuật toán

Tìm kiếm tuyến tính là bắt đầu từ phần tử đầu tiên, tìm một cách tuần tự trong danh sách cho đến khi tìm ra phần tử đã cho hoặc đạt đến cuối danh sách. Kiểu danh sách là mảng các phần tử có cấu trúc dữ liệu kiểu Kieu_Ptu.

Các khối chính của sơ đồ thuật toán được vẽ trên hình 8.1.

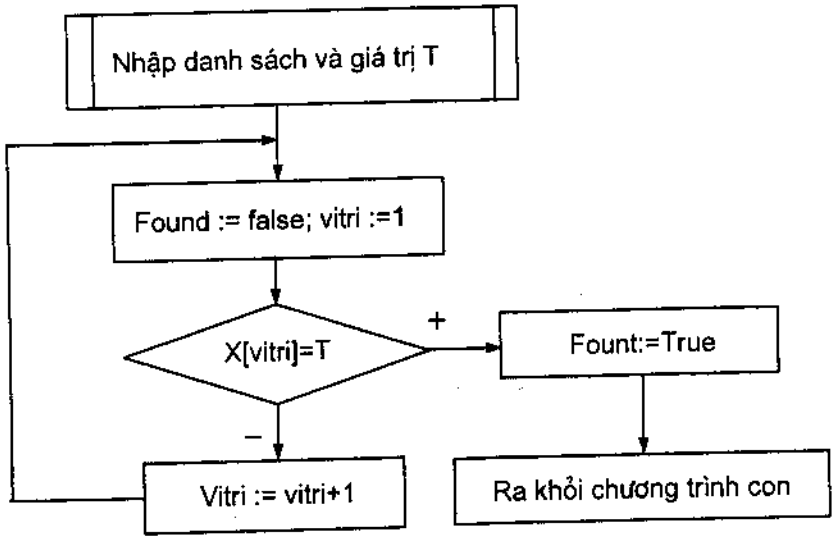
8.1.1.3. Chương trình con thủ tục

```
Procedure Timkiem1(var X: Kieu_Dsach; n: integer; vitri:integer;
                                   Item: Kieu_Ptu; var Found: boolean) ;
        Found:= false;
    Pos := 1;
    while  not  Found and (vitri <=n )  Do
       if  Item =  X[pos] then  Found := true
                                   else vitri := vitri +1
End;
```

8.1.2. Tìm trong một danh sách liên kết

Nếu là danh sách liên kết, ta dùng một con trỏ chỉ vị trí xử lý PosPtr.

```
Procedure Timkiem2(L: Listpointer; Item: Kieu_Ptu;
    Var Found: Boolean; var vitriPtr : Listpointer);
        Begin Found : = false ; vitriPtr : = List;
            while not found and vitriPtr < > Nil Do
             if vitriPtr^.Du_Lieu = Item then Found : = true
                else vitriPtr : = vitriPtr^.Next;
        end;
```



Hình 8.1

8.1.3. Tìm kiếm tuyến tính trong danh sách được sắp thứ tự

Cho danh sách có thứ tự tăng dần về giá trị $x_1, x_2, ..., x_n$ cần tìm phần tử trong danh sách đó.

1. Found: = false; DoneSearching := false; I = 1; Loc := 0;
2. While not DoneSearching Do
   if Item = X[i] then
      begin Found:= true; DoneSearching := true; Loc := i end
   else
      if Item < X[i] or i =n then
         begin Found: =true, Done Searching := true end
         Else i:= i+1

Trường hợp Item < x[1] hoặc Item> x[n], vòng lặp thoát ra với vị trí tìm thấy bằng Loc = 0.

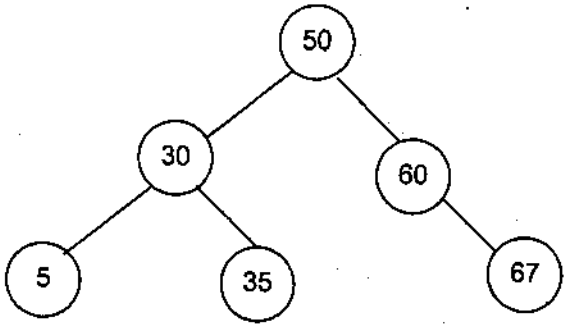
8.1.4. Tìm kiếm nhị phân

Tìm phần tử trong danh sách x_1, x_2, ..., x_n được sắp xếp theo thứ tự tăng dần. Found sẽ có giá trị đúng khi tìm được và Mid là vị trí của phần tử nếu tìm ra.

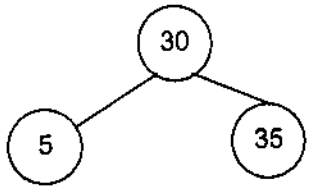
```
1. Found : = false, First: = 1; last: = n;
2. While First ≤Last and not Found then
      begin mid : = (First + last)/2;
            if Item < x[mid] then last:= Mid-1
                Else
            if Item > X[Mid] then First := Mid+1
                Else Found := true
      end;
```

8.2. CÂY TÌM KIẾM NHỊ PHÂN

Trong một cây nhị phân mà giá trị ở mỗi nút lớn hơn giá trị ở con bên trái của nó (nếu có) và nhỏ hơn giá trị ở con bên phải của nó thì được gọi là cây tìm kiếm nhị phân vì có thể dùng thuật toán gần giống tím kiếm nhị phân để tìm kiếm trên nó (Binary Search Tree- BST).



Giả sử muốn tìm phân tử 35. Ta bắt đầu từ nút gốc vì 35 < 50 → Giá trị cần tìm ở bên trái nút gốc → nghĩa là nó cây con phía trái.



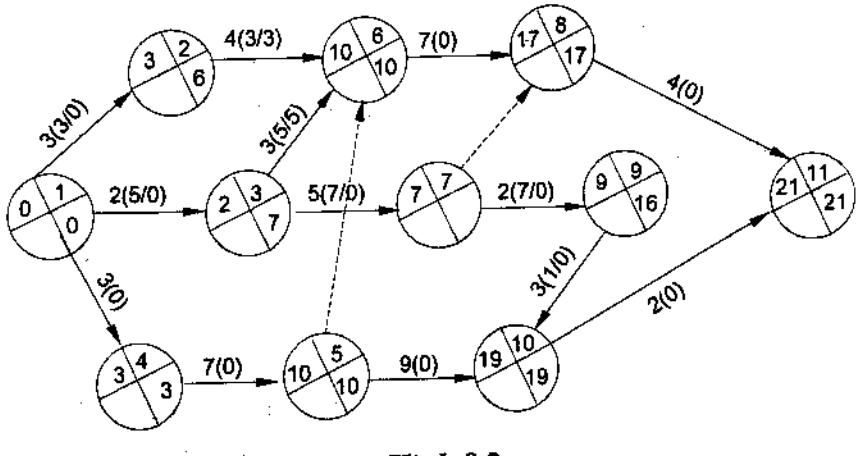
So sánh tiếp 35 với gốc cây con: vì 35 > 30 → ở phía cây con bên phải :

Nó là duy nhất → kết thúc.

Viết thủ tục tìm kiếm trên cây nhị phân: gốc - root, và nút tìm thấy - LocPtr là kiểu con trỏ, trỏ đến nút có phần Du_Lieu chứa cùng trường Id như trong Item (phần tử), Item thuộc kiểu Kieu_Ptu.

```
Procedure BSTSearch(Root : TreePointer ; Item : Kieu_Ptu);
        Var Found: Boolean; var LocPtr : TreePointer ;
Begin
            LocPtr : = Root ; Found: = false;
    While not found and (LocPtr < > nil) Do
            with LocPtr^ Do
    if Item.Id < Du_Lieu.Id then LocPtr: = Lchild
    else if Item.Id > Du_Lieu.Id then LocPtr: = Rchild
            else Found: = true
end;
```

8.3. TÌM LỜI GIẢI TỐI ƯU TRÊN SƠ ĐỒ MẠNG



Hình 8.2

8.3.1. Khái quát về sơ đồ mạng

Sơ đồ mạng được sử dụng để lập và điều khiển tiến độ thi công các dự án, từ những dự án xây dựng công trình đến cả dự án chế tạo tên lửa. Ngày nay, để giải quyết bất kỳ một nhiệm vụ phức tạp nào trong nghiên cứu khoa học, kỹ thuật, kinh tế, xã hội, an ninh, quốc phòng, người ta đều sử dụng Sơ đồ mạng như một công cụ quản lý hữu hiệu.

Các dự án thường có rất nhiều công việc. Muốn thực hiện dự án một cách khoa học, đúng tiến độ, đạt chất lượng cao, và giảm chi phí tới mức tối thiểu về công sức và tiền của bắt buộc ta phải có phương pháp quản lý tốt nhất. Phương pháp Sơ đồ mạng có thể giúp ta giải được bài toán này. Để tư duy theo phương pháp Sơ đồ mạng, ta phải biết chính xác các thông tin sau:

- Dự án cần bao nhiêu thời gian để hoàn thành.

- Thời điểm bắt đầu hoặc hoàn thành công việc, thời gian thực hiện từng công việc.

- Những công việc nào là trọng tâm, cần tập trung sức tài vật lực để hoàn thành đúng tiến độ.

Sơ đồ mạng được xây dựng trên mô hình toán học hiện đại, đó là Lý thuyết đồ thị với hai yếu tố logic cơ bản là: công việc và sự kiện. Trong sơ đồ mạng, các công việc được biểu diễn một cách sinh động và cụ thể. Không chỉ thấy tên công việc mà nó còn cho thấy mối liên hệ giữa công việc này với công việc khác. Để lập được sơ đồ mạng cần phân tích tỉ mỉ trình tự, công việc, những mối liên hệ bắt buộc về công nghệ hoặc logic về tổ chức.

Có nhiều phương pháp sơ đồ mạng, phổ biến nhất là phương pháp đường găng (Critical Path Methol - CPM) và phương pháp sơ đồ PERT (Program Evaluation and Review Technique). Sơ đồ mạng CPM gồm những đường và nút trong đó các nút là các sự kiện, các đường là các công việc; còn trong sơ đồ mạng PERT thì phần tử chính là các công việc, còn các đường chỉ là những mũi tên đơn thuần để liên hệ các nút là các công việc lại với nhau.

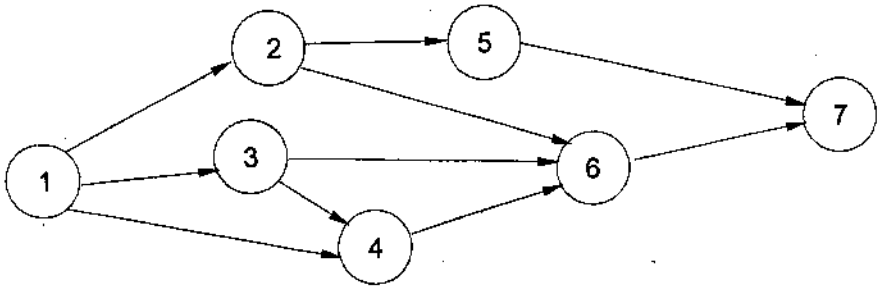
8.3.2. Công việc

Công việc ở đây thể hiện một quá trình nào đó, có thể chỉ là mối liên hệ phụ thuộc. Mỗi công việc được thể hiện bằng một mũi tên và được gọi

tên bằng ký hiệu của hai sự kiện trước và sau, các ký hiệu này về thứ tự là rất quan trọng vì nó chỉ định công việc đó bắt đầu từ sự kiện nào và kết thúc ở sự kiện nào. Có hai dạng công việc.

8.3.2.1. Công việc thực: Là công việc cần sự chi phối về thời gian và tài nguyên, nhân lực hoặc chỉ là thời gian chờ đợi, được thể hiện bằng một nét liền.

8.3.2.2. Công việc ảo: Công việc ảo là công việc chỉ thể hiện mối liên hệ logic giữa hai hoặc nhiều công việc, không đòi hỏi chi phí thời gian và vật tư, nhân lực. Trên sơ đồ, công việc ảo biểu diễn bởi nét đứt.



Hình 8.3

8.3.2.3. Các quan hệ: Là mối liên hệ giữa các việc với nhau. Đó chính là sự phụ thuộc lẫn nhau của các công việc trong quá trình thi công xây dựng.

8.3.2.4. Sự kiện

Sự kiện là cái mốc đánh giá sự bắt đầu một công việc, còn đi vào là hoàn thành công việc. Sự kiện đầu không có công việc đi vào gọi là sự kiện xuất phát (luôn ký hiệu là số 1). Sự kiện cuối cùng không có công việc đi ra là sự kiện kết thúc hay hoàn thành; Sự kiện hoàn thành luôn là sự kiện có chỉ số cao nhất trong tất cả các sự kiện.

8.3.2.5. Đường

Đường là một chuỗi các công việc được sắp xếp sao cho sự kiện cuối của công việc này là sự kiện đầu của công việc kia. Chiều dài của đường tính theo giá trị thời gian (ngày, giờ, tháng). Đường trong sơ đồ mạng bao giờ cũng đi từ điểm xuất phát đến điểm hoàn thành nên có rất nhiều đường như vậy. Đường có độ dài lớn nhất đi từ sự kiện đầu tiên đến sự kiện cuối cùng là *đường găng*.

8.3.2.6. Thời gian sử dụng cho công việc

Thời gian sử dụng cho công việc là khoảng thời gian để hoàn thành công việc. Thông thường thời gian sử dụng cho công việc được ký hiệu là t_{ij}.

8.3.2.7. Tài nguyên

Tài nguyên trong sơ đồ mạng được hiểu là thời gian và các vật tư cần thiết cho quá trình thực hiện dự án, như: Tiền vốn, công nhân, vật liệu, vật tư, trang thiết bị, máy móc.

8.3.3. Các thông số của sơ đồ mạng

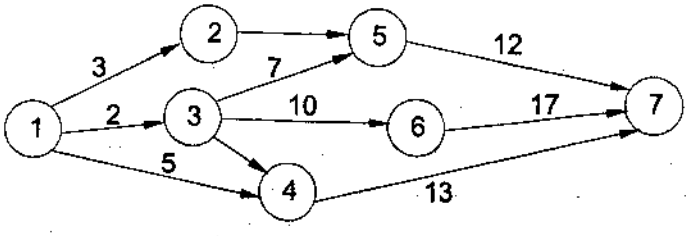
Khi mô hình mạng đã được thiết lập, người ta sẽ tính được thời gian của công việc trên cơ sở các định mức về lao động. Tuy nhiên, thời gian hoàn thành dự án là bao nhiêu thì phải có phương pháp tính toán để xác định. Muốn tính được điều đó cần phải biết các thông số sau:

- Thời điểm xuất hiện sớm và muộn của từng sự kiện (sau đây sẽ gọi tắt là thời gian sớm, thời gian muộn của sự kiện).
- Thời điểm khởi công sớm và khởi công muộn nhất của từng công việc.
- Thời điểm hoàn thành sớm và hoàn thành muộn nhất của từng công việc.
- Đường găng.

8.3.3.1. Thời gian sớm của sự kiện ($T_i^s T_j^s$)

Thời gian xuất hiện sớm nhất của một sự kiện i là thời điểm tính từ lúc khởi công đến khi xảy ra sự kiện thứ i sau khi đã hoàn thành tất cả các công việc đi trước i theo trình tự logic.

Ví dụ: Cho sơ đồ mạng với các tham số thời gian công việc ghi bên cạnh từng công việc (i, j) như trong hình 8.4



Hình 8.4

Sự kiện 5: Chỉ có thể xảy ra sớm nhất khi các công việc 1-2, 2-3, 1-3, đều đã hoàn thành, do đó thời gian sớm nhất của sự kiện 3 là 3 + 8 = 11 ngày.

Tương tự cho các sự kiện khác, nếu sự kiện i có nhiều con đường đi đến thì thời gian xuất hiện sớm của sự kiện (T_i^s) sẽ là "con đường lớn nhất đi từ sự kiện 1 đến sự kiện i".

Để thiết lập công thức tính toán ta dùng ký hiệu sau:

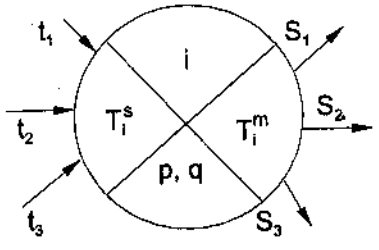
i: Sự kiện;

T_i^s: Thời gian sớm của sự kiện i;

T_i^m: Thời gian muộn của sự kiện i;

t_{ij}: Thời gian của công việc ij;

p, q: Sự kiện đi đến i có con đường dài nhất.



Hình 8.5

Các công việc trước khi đến sự kiện đang xét được ký hiệu là t_1, t_2, t_3 và các công việc sau khi đi ra từ sự kiện là S_1, S_2, S_3...

Chiều dài của mỗi đường đi trong sơ đồ mạng t(L) là tổng thời gian các công việc nằm trên đường đó, và được tính bằng công thức:

$$t(L) = \sum_i t_{ij}$$

Để tính thời gian xuất hiện sớm của sự kiện, ta bắt đầu từ sự kiện xuất phát của sơ đồ mạng, sau đó tính truy hồi qua các sự kiện cho đến sự kiện cuối cùng.

Trong quá trình trung gian, nếu ta đang ở sự kiện thứ i thì thời gian xuất hiện sớm của sự kiện j ngay sau đó sẽ được tính bằng công thức:

$$T_j^s = T_i^s + t_{ij}$$

(Tức là thời gian xuất hiện sớm của sự kiện j bằng thời gian xuất hiện sớm của sự kiện i trước nó cộng với thời gian thực hiện công việc (i, j).)

Nếu trước sự kiện j có nhiều sự kiện đi đến nó thì T_j^s sẽ bằng con đường lớn nhất từ các sự kiện đó tới j:

$$T_j^s = \max\{(T_i^s + t_{ij}); (T_h^s + t_{hij});...\}$$

Trong đó sự kiện nào có con đường lớn nhất đi qua nó đến j (và sự kiện phải đứng ngay trước nó) sẽ được ghi vào ô dưới vòng tròn sự kiện (ví dụ: p). Nếu có nhiều sự kiện đi đến j có con đường bằng nhau thì ta cũng ghi vào ô dưới các sự kiện đó.

8.3.3.2. Thời gian muộn của sự kiện (T_i^m ; T_j^m)

Một sự kiện chỉ được phép xảy ra muộn nhất vào ngày xác định để không làm ảnh hưởng đến thời gian hoàn thành dự án. Như vậy là để đảm bảo tiến độ, sự kiện cuối cùng là hoàn toàn xác định (nó chính là thời gian pháp lệnh của công trình, dự án), thời gian xuất hiện sớm và xuất hiện muộn của nó phải trùng nhau:

$$T_n^s = T_n^m$$

Việc tính thời điểm xuất hiện muộn của sự kiện sẽ được bắt đầu từ sự kiện cuối cùng, lùi về các sự kiện trước nó, theo cách tính: thời điểm xuất hiện muộn của sự kiện i sẽ bằng thời điểm xuất hiện muộn của sự kiện j sau nó trừ đi thời gian thực hiện công việc (i, j) ta thiết lập được công thức:

$$T_i^m = T_j^m - t_{ij}$$

Nếu đứng sau sự kiện i có nhiều sự kiện có thể lùi đến i thì T_i^m bằng hiệu số nhỏ nhất tính từ thời điểm muộn của các sự kiện lùi đó trừ đi thời gian công việc đó.

$$T_j^m = \text{Min}\{(T_i^m - t_{ij}); (T_h^m + t_{ij}); ...\}$$

8.3.3.3. Thời gian sớm của công việc (T_{ij}^{ks} ; T_i^{ks})

Một công việc có hai sự kiện đầu (i) và cuối (j) cho nên thời gian của công việc gắn với thời gian của sự kiện. Một công việc có hai thời điểm sớm là khởi sớm (hay gọi là bắt đầu sớm) ký hiệu T_{ij}^{ks} và hoàn thành sớm ký hiệu là T_{ij}^{hs}.

Ta có thể thấy rằng một công việc là khởi sớm nếu nó bắt đầu tại thời điểm sớm của sự kiện đầu:

$$T_{ij}^{ks} = T_i^s \text{ (tại sự kiện đầu i)}$$

và nó sẽ hoàn thành sớm nếu nó hoàn thành tại thời điểm sớm của sự kiện sau đó:

$$T_{ij}^{ks} = T_j^s \text{ (tại sự kiện sau:j) và } T_i^s = T_i^s + t_{ij}$$

nên: $T_{ij}^{ks} = T_{ij}^{ks} + t_{ij}$

Khi thời gian của một công việc không đổi thì nó sẽ hoàn thành sớm nhất nếu nó khởi công sớm nhất.

8.3.3.4. Thời gian muộn của công việc (T_{ij}^{km}; T_{ij}^{hm})

Một công việc có hai thời điểm muộn là khởi công muộn (bắt đầu muộn) ký hiệu là T_{ij}^{km} và hoàn thành muộn ký hiệu là T_{ij}^{km} tương ứng với thời điểm muộn của hai sự kiện đầu và cuối.

Cũng như thời gian sớm, một công việc là khởi muộn nếu nó bắt đầu vào thời điểm muộn của sự kiện đầu.

$$T_{ij}^{km} = T_i^m \text{ (tại sự kiện đầu: i)}$$

và là hoàn thành muộn nếu nó hoàn thành ở thời điểm muộn của sự kiện sau:

$$T_{ij}^{hm} = T_j^m \text{ (tại sự kiện sau: j)}$$

Tương tự khi thời gian công việc không đổi, ta có công thức:

$$T_{ij}^{km} = T_{ij}^{hm} - t_{ij}$$

8.3.3.5. Đường găng và ý nghĩa

Đường găng là đường từ điểm xuất phát đến điểm hoàn thành có chiều dài lớn nhất. Các công việc nằm trên đường găng là "công việc găng" trên sơ đồ mạng; các công việc găng được đánh dấu bằng cách vẽ màu đỏ hay tô đậm. Thời gian của đường găng cũng là thời gian hoàn thành dự án, hoặc thời gian xây dựng công trình. Trong sơ đồ mạng thường có một đường găng nhưng cũng có thể có nhiều đường găng.

Những công việc găng trên phải được hoàn thành đúng thời gian quy định, nghĩa là phải khởi và kết đúng thời điểm đã được xác định. Khi đã xác định được các công việc găng, người chỉ huy công trình cần phải tập

trung sức tài vật lực để hoàn thành các công việc găng đúng tiến độ, khi đó mới đảm bảo đúng tiến độ cho cả công trình.

Trong khi các công việc không găng có thể kéo dài thêm thời gian t_{ij} hoặc thay đổi thời điểm khởi kết công việc trong giới hạn thời gian nào đó. Các giới hạn này gọi là "dự trữ thời gian".

8.3.3.6. Thời gian dự trữ của sự kiện

Một sự kiện có hai thời điểm sớm và muộn. Nó không thể xuất hiện trước thời điểm sớm nhất (T_i^s) và cũng không thể xuất hiện sau thời điểm muộn nhất (T_i^m). Nhưng, nó cũng có thể xuất hiện bất kỳ lúc nào giữa hai thời điểm đó. Khoảng thời gian chênh lệch giữa thời hạn sớm và muộn là khoảng thời gian dự trữ của sự kiện. Đó chính là thời gian mà sự kiện có thể chậm lại mà không làm ảnh hưởng đến thời hạn hoàn thành dự án. Ta tính được dự trữ của sự kiện bằng công thức:

$$D_i = T_i^m - T_i^s$$

Nếu dự trữ của sự kiện i không thay đổi $D_i = 0$, thì ta gọi đó là sự kiện găng. Các công việc găng phải gồm hai sự kiện đầu và cuối đều găng. Nhưng đây chỉ là điều kiện cần mà chưa đủ, bởi vì nhiều khi nối hai sự kiện găng lại nhưng chưa chắc đã được một công việc găng. Tương tự, đường găng sẽ đi qua tất cả các sự kiện găng và ngược lại sẽ không đúng, nghĩa là nếu nối tất cả các sự kiện găng ta sẽ được một đường chưa chắc là đường găng.

8.3.3.7. Thời gian dự trữ của công việc

Một công việc có hai sự kiện. Mỗi sự kiện có hai thời điểm sớm và muộn. Vì vậy, công việc có bốn thời điểm: Khởi công sớm - hoàn thành sớm, khởi công muộn - hoàn thành muộn.

Tương tự, một công việc cũng có bốn loại dự trữ, mỗi loại dự trữ có liên quan tới bốn thời điểm nói trên.

a) Dự trữ toàn phần của công việc

Dự trữ lớn nhất (D_{ij}): còn gọi là dự trữ toàn phần là loại dự trữ về thời gian nếu ta sử dụng nó để thay đổi các thời điểm khởi sớm - kết sớm,

khởi muộn - kết muộn, hoặc kéo dài thời gian t_{ij} thì chỉ làm ảnh hưởng các công việc trước và sau nó mà không làm ảnh hưởng đến thời hạn hoàn thành dự án nó được tính theo công thức:

$$D_{ij} = T_j^m - T_i^s - t_{ij}$$

b) Dự trữ riêng của công việc

Dự trữ riêng hay còn gọi là dự trữ bé nhất (d_{ij}): là loại dự trữ thời gian nếu ta sử dụng sẽ không làm ảnh hưởng đến các công việc trước và sau nó, được tính theo công thức:

$$d_{ij} = T_j^s - T_i^m - t_{ij}$$

Công việc nào có dự trữ toàn phần = 0 và dự trữ riêng t = 0 là công việc găng.

c) Dự trữ do khởi sớm

Dự trữ do khởi sớm (d_{ij}^{ks}), còn gọi là dự trữ phụ thuộc: là dự trữ khi sử dụng sẽ tự do khởi sớm hoặc kéo dài thời gian công việc mà không ảnh hưởng đến sự khởi sớm của công việc tiếp theo:

$$d_{ij}^{ks} = T_j^s - T_i^s - t_{ij}$$

d) Dự trữ do khởi muộn

Dự trữ do khởi muộn (d_{ij}^{km}) là sự trữ liên quan, là dự trữ khi sử dụng sẽ tự do khởi muộn và kéo dài thời gian công việc mà không ảnh hưởng đến sự khởi muộn tiếp theo:

$$d_{ij}^{km} = T_j^m - T_i^m - t_{ij}$$

8.3.4. Các phương pháp tính sơ đồ mạng

Hiện nay có 2 phương pháp cơ bản để tính sơ đồ mạng đó là: tính sơ đồ mạng trực tiếp trên sự kiện: tính sơ đồ mạng bằng lập bảng.

8.3.4.1. Tính sơ đồ mạng trực tiếp trên sự kiện

Theo phương pháp này người ta chia sự kiện ra làm 4 ô. Sự kiện được biểu diễn bằng đường tròn nên có tên gọi là "vòng tròn sự kiện". Các thông số được ký hiệu như sau:

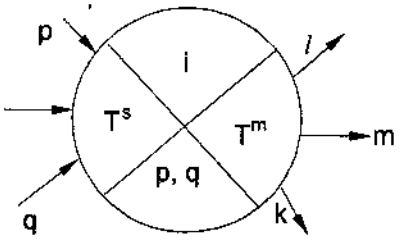
i: Sự kiện đang xét;

T^s: Thời gian sớm của sự kiện i;

T^m: Thời gian muộn sớm của sự kiện i; i

t_{ij}: Thời gian của công việc ij;

p, q: Sự kiện đi đến i có đường dài nhất.



Hình 8.6

Xét ví dụ đã cho trong hình 8.3.

Trình tự tính toán theo 3 bước như sau:

Bước 1: Lượt đi tính từ trái sang phải

Tính thời điểm sớm của sự kiện (T^s)

- Bắt đầu từ sự kiện xuất phát tới $T_1^s = 0$

- Sự kiện tiếp theo nếu chỉ có một công việc đi đến sẽ tính theo công thức.

$$T_j^s = T_i^s + t_{ij}$$

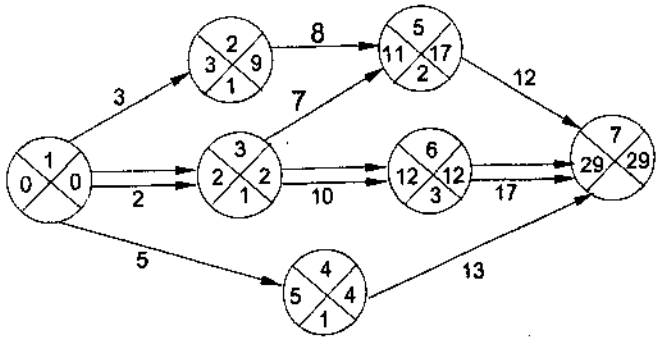
Nếu có công việc đi đến sẽ tính như sau:

$$T_j^s = \max [(T_i^s + t_{ij}); (T_h^s + t_{hi}); ...]$$

- Sự kiện nào đứng trước mà đi đến sự kiện đang xét bằng con đường dài nhất sẽ được ghi ở ô dưới (Nếu có hai hoặc nhiều sự kiện đứng trước đi đến sự kiện đang xét đều có chiều dài bằng nhau, sẽ được ghi tất cả vào ô dưới)

- Cứ như vậy tính dần lên theo thứ tự tăng dần của chỉ số sự kiện, cho đến sự kiện hoàn thành cuối cùng (T_n^s) thì sẽ hoàn thành bước thứ nhất.

Kết quả bước thứ nhất tính vào ô trái của sự kiện (,.) và các chỉ số ở ô dưới sự kiện.



Hình 8.7

Bước 2: Tính thời điểm muộn của sự kiện (T^m)

- Bắt đầu từ sự kiện cuối cùng $T_n^m = T_n^s$

Nghĩa là dù sớm hay muộn cũng phải hoàn thành kế hoạch tiến độ đúng thời hạn. Do đó, thời điểm sớm hay muộn của sự kiện cuối cùng bằng nhau.

- Tính ngược lại sự kiện (n-1), (n-2), ..., i, ..., 1.

Ta có công thức:

$$T_i^m = T_j^m - t_{ij}$$

Nếu có nhiều sự kiện đứng sau sự kiện đang xét i có thể lùi đến sự kiện i bằng nhiều công việc thì T_i^m được tính bằng công thức:

$$T_i^m = \text{Min}\{(T_i^m - t_{ij}); (T_k^m + t_{ik}); ...\}$$

- Cứ như vậy tính lùi về sự kiện xuất phát số 1 ta kết thúc bước 2. Kết quả bước 2 tính được ô phải của sự kiện T^m.

Bước 3: Xác định đường găng

Điều kiện cần và đủ của đường găng là đường đi qua các sự kiện găng và là đường dài nhất.

Một cách đơn giản để xác định các sự kiện găng là các sự kiện có dự trữ $D_i = T_i^m - T_i^s = 0$; nghĩa là ô trái và ô phải của sự kiện bằng nhau. Ngoài ra, còn phải xét xem đường nối các sự kiện găng có phải là đường dài nhất không. Trong ví dụ ở hình 8.7, các công việc găng nối các sự kiện găng được tô đậm.

8.3.4.2. Tính sơ đồ mạng bằng phương pháp lập bảng

Theo phương pháp này ta lập bảng để tính toán các thông số thời gian: khởi sớm, kết sớm, khởi muộn, kết muộn của từng công việc. Tính dự trữ toàn phần và dự trữ riêng của từng công việc, qua đó ta cũng có thể xác định đường găng. Trình tự tiến hành tính toán theo các bước sau:

Bước 1: Lập bảng tính toán

Các công việc được sắp xếp theo thứ tự tăng dần của sự kiện đầu và cuối của sự kiện.

TT	Tên công việc	Ký hiệu (ji)	Thời gian t_{ij} (ngày)	Nhân công (người)	Sớm		Muộn		Dự trữ		Công việc găng
					T_{ij}^{ks}	T_{ij}^{hs}	T_{ij}^{km}	T_{ij}^{hm}	D_{ij}	d_{ij}	
1	2	2	4	5	6	7	8	9	10	11	12

Bước 2: Tính thời gian khởi sớm của công việc:

Với điều kiện các công việc đều là khởi sớm, tức là khởi công tại thời điểm sớm của sự kiện đầu:

$$T_{ij}^{ks} = T_{ik}^{ks} = T_{ih}^{ks} = ...T_i^s$$

Bước 3: Tính thời gian hoàn thành muộn của công việc:

$$T_{ij}^{hm} = T_{hj}^{hm} = T_{gj}^{hm} = ...T_j^s$$

Bước 4: Tính trực tiếp trên bảng tính. Tính thời gian kết sớm của công việc theo công thức sau và ghi vào bảng (cột 7):

$$T_{ij}^{hs} = T_{ij}^{hs} + t_{ij}$$

Tính lùi thời gian khởi công muộn công việc theo công thức sau và ghi vào bảng (cột 8):

$$T_{ij}^{km} = T_{ij}^{hm} - t_{ij}$$

Tính dự trữ lớn nhất D_{ij} của công việc và ghi vào bảng (cột 10):

$$D_{ij} = T_j^m - T_i^s - t_{ij}$$

Tính dự trữ bé nhất d_{ij} của công việc và ghi vào bảng(cột 11):

$$d_{ij} = T_j^s - T_i^m - t_{ij}$$

8.3.5. Các thuật toán

Cho đến nay ở trên thế giới đã có nhiều thuật toán và chương trình về sơ đồ mạng (SĐM). Ở nước ta một số nơi cũng đã có các chương trình về SĐM, song việc khai thác sử dụng chung có nhiều hạn chế. Để phục vụ cho việc tính toán kế hoạch và quản lý thi công trong xây dựng chúng sẽ khảo sát thuật toán tính các tham số của SĐM: thuật toán SĐM.01 - tính tham số thời gian của đỉnh, SĐM.02 - tính các tham số thời gian công việc.

8.3.5.1. Thuật toán SĐM.01

Trong quá trình xây dựng công trình được mô tả dưới dạng một SĐM là một đồ thị có hướng, không có quay vòng G(D, T), trong đó D = $\{d_i\}$ là tập các đỉnh - sự kiện, mỗi công tác xây dựng được đặc trưng bởi sự kiện khởi đầu i và sự kiện kết thúc j ; T = t_{ij} là tập các thời gian công việc. yêu cầu phải tính thời điểm xuất hiện sớm T_i và thời điểm xuất hiện muộn T_i^m, dự trữ thời gian d_i của mỗi đỉnh, vạch ra đường găng của SĐM. Thuật toán gồm các bước sau:

Giai đoạn 1: Quá trình thuận.

1. Mô tả các tập sự kiện, công việc.

2. Gán số liệu ban đầu cho các tham số thời gian

3. Cho i bắt đầu từ 2, thiết lập đỉnh D_i

$$D_i = \{d_j | (j < i) \wedge (t_{ij} > 0) \wedge (T_j^s > 0)\}$$

4. $$T_i^s = \begin{cases} \max\limits_{j \in D} \left(T_i^s + t_{ij}\right) \text{ nếu } D_i \neq \phi \\ 0 \text{ nếu } D_i' = \phi \end{cases}$$

5. Nếu i ≤ n thì tăng lên 1 đơn vị và lặp lại từ bước 3, nếu i > n thì chuyển sang giai đoạn sau.

Giai đoạn 2: Quá trình ngược.

1. $T_i^m = T_i^s$, i = N,

2. i = i-1, tập $D_i = \{d_j \mid (j > i) \wedge (t_{ij} > 0) \wedge (T_j^m > 0)\}$,

3. $T_i^m = \begin{cases} \min\limits_{j \in D} (T_j^m - t_{ij}) \text{ nếu } D_i \neq 0 \\ 0 \text{ nếu } D_i = \phi \end{cases}$

4. Nếu i > 1 thì lặp lại bước 2, ngược lại thì chuyển sang giai đoạn 3.

Giai đoạn 3: Tính dự trữ thời gian của sự kiện.

$$DT_i = (T_i^n - T_i^s (\forall i = \overline{i, N})$$

Giai đoạn 4: Đưa ra các công việc găng

$$SKG = \{d_i \mid DT_i = 0\}$$

8.3.5.2. Thuật toán SĐM.02

Ký hiệu: T_{ij}^{ks} là thời điểm khởi công sớm công việc ij; các kí hiệu T_{ij}^{km}, T_{ij}^{hs}, T_{ij}^{hm} là thời điểm khởi công muộn, hoàn thành sớm, hoàn thành muộn của công việc ij;

R_{ij}, r_{ij} là dự trữ thời gian toàn bộ, thời gian tự do của công việc ij.

Giai đoạn 1: Quá trình thuận.

1. Tạo các tập sự kiện, công việc, gán các giá trị ban đầu

2. $i = 1, t_{ij}^{ks} = 0 \quad \forall j > 1$ và $t_{ij} \neq 0$

3. $t_{ij}^{hs} = t_{ij}^{ks} + t_{ij}$

4. i = i +1, lập $CV_{ij} = \{(i, j) \mid T_{ij}^{hs} = T_{ij}^{ks} = 0\}$

$$CVTG_{ki}\{(k, i) \mid T_{ki}^{hs} > 0,\ T_{ki}^{hs} > 0\}$$

5. $T_{ij}^{ks} = T_i^m = \begin{cases} \max T_{ij}^{hs} \text{ nếu } CVGT_{ki} \neq \phi \\ 0 \text{ nếu } CVGT_{ki} = \phi \end{cases}$

6. Nếu i > N thì lặp lại từ bước 3, ngược lại chuyển sang giai đoạn 2.

Giai đoạn 2: Quá trình ngược.

1. $i = N,\ T_{ij}^{hm} = t_{găng}$,

2. $T_{ij}^{km} = T_{ij}^{hm} - t_{ij}$

3. $J = j\text{-}1,\ CV_{ij} = \{(i, j) |\ T_{ij}^{km} = T_{ij}^{hm} = 0\}$ lập $CVGT_{jk} = \{(j, k) | T_{jk}^{hm} > 0, T_{jk}^{km} > 0\}$,

4. $T_i^m = \begin{cases} \min T_{jk}^{km} \text{ nếu } CVTG_{jk} \neq \phi \\ 0 \text{ nếu } CVTG_{jk} = \phi \end{cases}$

5. Lặp lại từ bước 3 cho đến khi $j = 1$

Giai đoạn 3: Tính dự trữ toàn phần và dự trữ tự do.

$$R_{ij} = T_{ij}^{hm} - T_{ij}^{hs},\ r_{ij} = \max\{T_{kj}^{ks}\} - T_{ij}^{ks}.$$

Chương 9

QUY HOẠCH ĐỘNG

9.1. KHÁI NIỆN CHUNG

9.1.1. Mở đầu

Ngày nay quy hoạch toán học đã đóng một vai trò quan trọng trong các ngành kĩ thuật. Sự phát triển như vũ bão của kĩ thuật tính toán đã thúc đẩy việc nghiên cứu phát triển thêm nhiều môn quy hoạch toán học mới. Với sự giúp sức của máy tính điện tử (MTĐT), một số phương pháp quy hoạch toán học đã cho phép giải quyết được những bài toán mà đối với phương pháp giải tích cổ điển xem như bất lực.

Một trong những nét đặc trưng cơ bản của cuộc cách mạng khoa học - kĩ thuật hiện đại là việc toán học hoá. Toán học ứng dụng đã trở thành nhu cầu cấp thiết. Nếu chú ý vào việc xuất bản sách và tài liệu nước ngoài ta thấy các công trình nghiên cứu về lí thuyết, về ứng dụng các phương pháp toán quy hoạch ngày càng nhiều. Việc giảng dạy các môn toán quy hoạch trong các trường đại học đã trở thành phổ biến, ngoài những chương trình toán cao cấp cơ sở. Ở nước ta công việc đó đã bắt đầu từ sớm và môn quy hoạch toán học phổ biến hơn cả là quy hoạch tuyến tính. Một số môn quy hoạch toán học khác còn chưa được phổ biến lắm như quy hoạch phi tuyến, quy hoạch hình học, quy hoạch động v.v... Trong kĩ thuật thì lại thường gặp những bài toán không giải được bằng toán học tuyến tính, hoặc có thể giải được nhưng bất lợi hơn so với các phương pháp quy hoạch khác. Vì vậy việc nghiên cứu áp dụng chúng là rất cần thiết.

Từ những năm trước đây, thuật ngữ "tối ưu hóa" được khá phổ biến. Các phương pháp toán quy hoạch nói chung đều là những phương pháp tìm lời giải tối ưu, hay "phương án tối ưu". Thời gian đầu được coi như những thành viên của vận trù học, ngày nay các phương pháp quy hoạch

toán học đã được phát triển thành từng môn riêng biệt. Quy hoạch động cũng nằm trong tình trạng chung đó. Quy hoạch động do Giáo sư Bellman. R đề xướng và phát triển vào những năm 50 của thế kỉ này. Tuy còn mới mẻ nhưng quy hoạch động đã chiếm được ưu thế trong một lớp bài toán rộng lớn. Nó được sử dụng để giải quyết một số bài toán trong các quá trình hoá - kĩ thuật, trong các ngành kĩ thuật và kinh tế.

Richard Bellman (1920 - 1984) là một nhà toán học ứng dụng, Ông đã có nhiều đóng góp trong lĩnh vực toán học, nhưng đóng góp lớn nhất là đề xướng Quy hoạch động. Ông là Giáo sư Trường đại học Nam California, là thành viên Viện Hàn lâm Khoa học và Nghệ thuật Hoa kỳ (1975), đã từng được tặng thưởng Huy chương danh dự của IEEE năm 1979.

9.1.2. Định nghĩa Quy hoạch động

Quy hoạch động là một phương pháp cho phép giải các bài toán tối ưu mà có thể chia thành các quá trình nhiều bước, lời giải sẽ nhận được trên từng bước để cho cả quá trình là một phương án tối ưu.

Bản chất của quy hoạch động là trong một quá trình nhiều bước, lời giải nhận được ở bước cuối cùng là tối ưu ứng với mọi khả năng, mọi trạng thái của hệ ở trên bước thứ hai cuối cùng và nó cũng sẽ là tối ưu cho cả quá trình.

Như vậy muốn dùng quy hoạch động, việc đầu tiên là phải đưa bài toán về dạng một quá trình nhiều bước và sử dụng cách lập luận của quy hoạch động (sẽ nói sau), để tìm lời giải trên từng bước sao cho hàm mục tiêu đều đạt giá trị tối ưu (cực đại hay cực tiểu). Sau này ta sẽ thấy quy hoạch động thuộc vào những phương pháp tìm cực trị.

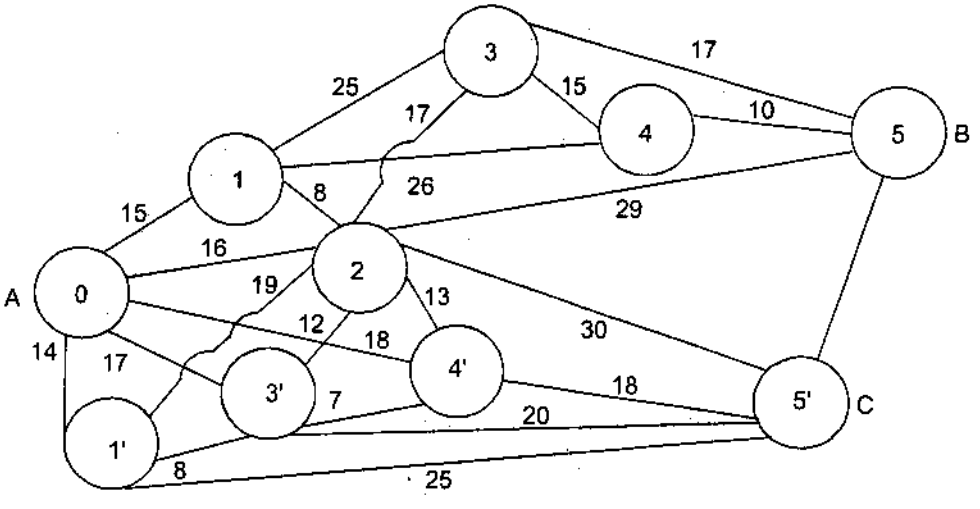
9.1.3. Ví dụ

Ở đây ta sẽ xét một ví dụ đơn giản để làm quen với một phương pháp suy luận của Quy hoạch động.

Ta có bài toán như sau:

Tại công trường xây dựng A phải sử dụng các loại vật liệu có thể lấy ở kho B và C. Hệ thống đường sá giữa A, B, C được chỉ trên hình vẽ, giả thiết là chi phí chuyên chở vật liệu đã được quy về giá trị trung

bình, ghi bên cạnh các tuyến đường. Bài toán yêu cầu chọn kho lấy vật liệu và đường đi sao cho tổng số chi phí là nhỏ nhất. Điều kiện ràng buộc ở đây là chỉ có thể đi từ điểm nút thứ i đến k, $i < k$, chứ không được ngược lại. Nếu dùng phương pháp tổ hợp thì bằng cách tính tất cả các phương án, ta cũng tìm ra được phương án tối ưu, vì ở đây số phương án không nhiều lắm. Nhưng ta sẽ dùng quy hoạch động và chỉ ra tính ưu việt của nó trong trường hợp số phương án khá lớn. Từ hình vẽ ta thấy việc chia từng bước ở đây được hình thành trên việc xét từng điểm mút, và không nhất thiết phương án nào (tuyến đường nào) cũng phải đi qua 5 nút theo thứ tự 5 - 4 - 3 - 2 - 1, ta gọi chi phí trên đoạn đường từ i tới j là G_{ij}; chi phí trên tuyến đường từ j đến B (C) theo phương án tối ưu là F_j (F'_j).



Hình 9.1

Từ cách định nghĩa trên ta có.

$$F_i = \min_j (G_{ij} + F_j); \quad F_{i'} = \min_{j'} (G_{i'j'} + F_{j'})$$

Thứ tự ta làm như sau:

Bước 1: $\begin{cases} F_5 = 0 \\ F'_5 = 0 \end{cases}$

Bước 2: $F_4 = \min_{5'} (G_{4,5'} + F_5) = \min (10 + 0) = 10$

$F'_{4'} = \min_{5'} (G_{4',5'} + F'_{5'}) = \min (18 + 0) = 18$

Bước 3:

$$F_3 = \min \begin{Bmatrix} G_{3,5} + F_5 \\ G_{3,4} + F_4 \end{Bmatrix} = \min \begin{Bmatrix} 17 + 0 \\ 15 + 10 \end{Bmatrix} = 17$$

$$F'_{3'} = \min \begin{Bmatrix} G_{3',5'} + F'_{5'} \\ G_{3',4'} + F'_{4'} \end{Bmatrix} = \min \begin{Bmatrix} 20 + 0 \\ 7 + 18 \end{Bmatrix} = 20$$

Bước 4:

$$F_2 = \min \begin{Bmatrix} G_{2,5} + F_5 \\ G_{2,3} + F_3 \end{Bmatrix} = \min \begin{Bmatrix} 29 + 0 \\ 17 + 17 \end{Bmatrix} = 29$$

$$F'_{2'} = \min \begin{Bmatrix} G_{2,5'} + F'_5 \\ G_{2,3'} + F'_{3'} \\ G_{2,4'} + F'_{4'} \end{Bmatrix} = \min \begin{Bmatrix} 30 + 0 \\ 12 + 20 \\ 13 + 18 \end{Bmatrix} = 30$$

Bước 5:

$$F_1 = \min \begin{Bmatrix} G_{1,3'} + F_3 \\ G_{1,4'} + F_4 \\ G_{1,2'} + F_2 \end{Bmatrix} = \min \begin{Bmatrix} 25 + 17 \\ 26 + 10 \\ 8 + 29 \end{Bmatrix} = 36$$

$$F'_{1'} = \min \begin{Bmatrix} G_{1',2'} + F_{2'} \\ G_{1',3'} + F'_{3'} \\ G_{1',5'} + F'_{5'} \end{Bmatrix} = \min \begin{Bmatrix} 19 + 30 \\ 8 + 20 \\ 25 + 0 \end{Bmatrix} = 25$$

Bước 6:

$$F_0 = \min \begin{Bmatrix} G_{0,1} + F_1 \\ G_{0,2} + F_2 \end{Bmatrix} = \min \begin{Bmatrix} 15 + 36 \\ 16 + 29 \end{Bmatrix} = 45$$

$$F'_0 = \min \begin{Bmatrix} G_{0,1'} + F'_{1'} \\ G_{0,2} + F_{2'} \\ G_{0,3'} + F'_{3'} \\ G_{0,4'} + F'_{4'} \end{Bmatrix} = \min \begin{Bmatrix} 14 + 25 \\ 16 + 30 \\ 17 + 20 \\ 18 + 18 \end{Bmatrix} = 36$$

Bước 7:

$$F_A = \min \begin{Bmatrix} F_0 \\ F'_0 \end{Bmatrix} = \min \begin{Bmatrix} 45 \\ 36 \end{Bmatrix} = 36$$

Như vậy, kết thúc phần tính toán ta thu được giá trị cực tiểu của hàm mục tiêu là 36 và đường tối ưu là A - 4'- C.

Điều cần chú ý ở đây là từ mỗi điểm mút đến các điểm cuối cùng (B, C), với việc tính giá trị cực tiểu ta còn xác định được những điểm mút mà đường tối ưu đi qua. Nếu ta tính bằng tay thì vị trí của chúng ta phải nhớ lấy hoặc ghi ra giấy, còn trong MTĐT thì phải giữ chúng lại trong bộ nhớ. Khi đã tính được giá trị cực tiểu của hàm mục tiêu từ điểm A, việc lần trở lại các điểm mút của đường tối ưu sẽ được thực hiện bằng một thủ tục, có thể giải trên MTĐT. Nếu giải bài toán trên bằng phương pháp tổ hợp thì sẽ rất phức tạp (gồm trên 20 phương án). Trong khi đó dùng quy hoạch động chỉ có 7 bước so sánh và lời giải nhận được đảm bảo là tối ưu.

9.2. NGUYÊN LÝ TỐI ƯU BELLMAN

9.2.1. Một số định nghĩa

Trong thí dụ ở chương trước, ta đã dùng một số khái niệm của toán điều khiển mà chưa định nghĩa, giải thích. Để tiện sử dụng trong các phần sau, ta sẽ bổ sung các định nghĩa của những khái niệm đó.

- ***Hệ:*** hệ là một khái niệm quan trọng và được sử dụng rộng khắp trong các ngành khoa học hiện đại. Có lẽ vì vậy, cho đến nay chưa có một định nghĩa nào được coi là duy nhất chấp nhận được. Một trong những định nghĩa được sử dụng nhiều trong toán điều khiển có thể phát biểu như sau: "Hệ là một tập hợp các tham số có một mối quan hệ tác động nào đó và

có một tính chất nào đó". Xét hệ bị điều khiển S, phương trình chuyển động có dạng:

$$\frac{dx}{dt} = f(x,u,t) \tag{9.1}$$

Trong đó x, u, f là các véctơ:

$f = (f^1, f^2, ..., f^n)$

$x = (x^1, x^2, ..., x^n)$

$u = (u^1, u^2, ..., u^m)$

Ở đây véctơ x thuộc không gian Ơcơlit n chiều E^n được gọi là véctơ trạng thái hay là tham số trạng thái. Véctơ u thuộc E^m được gọi là véctơ điều khiển hay là tham số điều khiển. Trong trường hợp tổng quát, *Tham số trạng thái* đặc trưng cho sự chuyển dời vị trí của hệ và cho trạng thái của hệ.

Véctơ điều khiển u(t) được chọn trong quá trình giải để cho hệ đạt được trạng thái x(t, u) với một tính chất nào đó. Trong lí thuyết điều khiển tối ưu, thường người ta xét tập hợp {u} các véctơ điều khiển là tập hợp các hàm liên tục từng đoạn. Giả sử hệ chuyển động từ vị trí X_T với thời gian T đã biết. Ta gọi *Hàm mục tiêu* là công thức diễn tả mục đích khảo sát hệ. Đối với hệ S trên, hàm mục tiêu có thể được cho bằng phiến hàm:

$$I(u) = \int_0^T f^0(x(t),u(t)dt) \tag{9.2}$$

Yêu cầu bài toán là chọn điều khiển u(t) thoả mãn phương trình chuyển động (1) đồng thời cho phiến hàm (2) giá trị cực tiểu. Trong thực tế, các véctơ x, u thường chỉ giới hạn xét trong một mối ràng buộc nào đó. Tập hợp những véctơ điều khiển u thoả mãn ràng buộc của bài toán làm thành tập hợp cho phép U. Đối với x cũng vậy.

Như vậy, trong một quá trình điều khiển gồm các thành phần:

- Phương trình chuyển động
- Tham số điều khiển
- Tham số trạng thái

- Các ràng buộc

- Hàm mục tiêu

Trong bài toán quy hoạch động, hàm mục tiêu phải có dạng hàm cộng được. Các ràng buộc đối với quy hoạch động không phải là nan giải, chúng có thể là dạng bất kì.

- Phương án hay chiến lược

Một phương án là một cách chọn các tham số điều khiển hệ, ứng với mỗi phương án hệ sẽ có một trạng thái xác định và hàm mục tiêu có một giá trị xác định. Phương án gọi là chấp nhận được khi các tham số nhận giá trị trong miền cho phép. Phương án tối ưu là một phương án cho phép vì nó cho hàm mục tiêu giá trị tối ưu. Trong bài toán quy hoạch động, việc chọn tham số nào làm tham số điều khiển là quan trọng, chọn đúng thì sẽ làm cho việc giải dễ dàng hơn rất nhiều.

9.2.2. Nguyên lí tối ưu

Phương pháp quy hoạch động dựa trên nguyên lí cơ bản sau đây, gọi là nguyên lí tối ưu:

"Một phương án tối ưu có tính chất như sau: Không phụ thuộc vào trạng thái ban đầu và điều khiển ban đầu, điều khiển tiếp theo cũng phải tạo thành lời giải tối ưu ứng với trạng thái sinh ra do kết quả tác động của điều khiển ban đầu".

9.2.3. Giải thích nguyên lí

Trước khi viết phương trình toán học cho nguyên lí, ở đây ta sẽ giải thích ý nghĩa của nó. Sử dụng nguyên lí tối ưu đảm bảo lời giải nhận được trên từng đoạn sẽ làm thành phương án tối ưu cho cả quá trình.

Trong nguyên lí, mệnh đề "Không phụ thuộc vào trạng thái ban đầu và điều khiển ban đầu" ta hiểu là không phụ thuộc vào quá khứ của quá trình. Như vậy là, tại bất kì một thời điểm nào của quá trình việc xét tối ưu tiếp theo chỉ phụ thuộc vào trạng thái của hệ tại điểm đó và hàm mục tiêu chứ không phụ thuộc vào những gì trước đó. Cụ thể hơn, nếu tại thời điểm t hệ đang ở trạng thái x(t) thì việc chọn điều khiển u(t) đưa hệ từ trạng thái x(t) sang x(t+1) chỉ phụ thuộc vào x(t) chứ không phụ thuộc vào x(t-1), x(t-2), v.v... ứng với mỗi trạng thái x(t) là kết quả tác động của

u(t-1) tương ứng, ta sẽ chọn được một điều khiển u(t) đưa hệ từ x(t) tới x(t+1) và cho hàm mục tiêu một giá trị tối ưu.

9.2.4. Quy hoạch động và lời giải thống kê

Phương pháp phân tích liên tục hay chọn lời giải liên tục các bài toán thống kê đã được A. Wald nghiên cứu ở Mĩ từ những năm cuối 1940.

Một quá trình lựa chọn liên tục được xét như sau: Trong một bài toán thống kê, nhà thống kê phải tiến hành các khảo sát X_1, X_2 ,..., với một phân bố xác suất nào đó phụ thuộc vào một tham số W chưa biết. Sau mỗi khảo sát X_n nhà thống kê có thể đánh giá thông tin về W, thu được trên cơ sở khảo sát X_1, X_2, ..., X_n. Trên cơ sở đó nhà thống kê sẽ quyết định khảo sát tiếp X_{n+1} hay không tiếp tục nữa.

R.Bellman và các học trò của ông đã phát triển một hướng của phương án phân tích liên tục, đề xướng ra quy hoạch động. Ở Liên Xô, Mikhalevitre V.C và các học trò của ông cũng phát triển phương pháp của Wald, lập nên "Sự phân tích liên tục các phương án" đã được sử dụng cuối những năm 50. Qua thực tiễn áp dụng cho thấy quy hoạch động cho phép giải quyết được cả một lớp bài toán rộng lớn, bao gồm cả những quá trình ngẫu nhiên và không ngẫu nhiên.

9.2.5. Phương trình hàm Bellman

9.2.5.1. Bài toán rời rạc

Phương trình hàm Bellman là phương trình cơ bản trong quy hoạch động. Để viết phương trình hàm ta xét một bài toán cụ thể như sau:

Giả sử có một số vốn x_t đầu tư vào việc mua thiết bị để sản xuất hai mặt hàng A, B trong một khoảng thời gian là T năm. Giả thiết quá trình sản xuất không có tái sản xuất. Trong năm đầu tiên, số thiết bị với vốn là u_t được dành sản xuất mặt hàng A, và $x_1 - u_1$ dùng cho sản xuất mặt hàng B. Cuối năm số lãi của từng loại vốn được cho bằng các hàm xác định $g(u_1)$ và $h(x_1-u_1)$ tương ứng. Sau một năm làm việc, giá trị của thiết bị với vốn u_t dùng để sản xuất mặt hàng A chỉ còn lại au_1, $0 \le a < 1$; tương tự đối với mặt hàng B giá trị thiết bị chỉ còn lại $b(x_1 - u_1)$, $0 \le b < 1$.

Như vậy, cuối năm thứ nhất, đầu năm thứ hai số vốn để mua thiết bị sản xuất trong năm tới này còn:

$$x_2 = au_1 + b(x_1 - u_1) \tag{9.3}$$

Quá trình phân phối vốn trên năm thứ hai và những năm sau cũng giống như trên năm thứ nhất, khi nào xét hết T năm thì quá trình kết thúc. Yêu cầu bài toán là hãy phân phối vốn trên các năm sao cho tổng số lãi trong cả quá trình T năm là cực đại, hay nói cách khác là chọn các điều khiển u_i sao cho hàm lợi nhuận đạt giá trị lớn nhất. Ta thấy rằng nếu trên năm đầu phương án đã chọn là tối ưu thì hàm mục tiêu chỉ phụ thuộc vào vốn ban đầu x_1, tức là:

$$f_1(x_1) = \max_{0 \le x_1 \le u_1} = [g(u_1) + h(x_1 - u_1)] \tag{9.4}$$

Ta dễ dàng nhận thấy rằng trong một qua trình I năm mới vốn ban đầu x, nếu phương án đã chọn là tối ưu thì hàm mục tiêu chỉ phụ thuộc vào x và I ta có định nghĩa:

$f_i(x)$ là lợi nhuận tối đa thu được trong quá trình I năm, với vốn ban đầu x và phương án đã chọn là tối ưu.

Quá trình ta xét là I năm, năm đầu ta phân phối vốn ra làm hai phần u_1 và $x_1 - u_1$. Như vậy để I - 1 năm còn lại ta có vốn đầu là: $x_2 = au_1 + (x_1 - u_1)$, và lợi nhuận thu được trong qua trình I năm bằng tổng lợi nhuận của năm đầu và của I – 1 năm sau:

$$g(u_1) + h(x_1 - u_1) + f_{i-1}(au_1 + (x_1 - u_1)).$$

Ứng với mỗi điều khiển u_1, giá trị $f_{i-1}(au_1 + (x_1 - u_1)$ và lợi nhuận tối đa trên I – 1 năm sau đó và tổng trên cho một giá trị xác định. Vấn đề là phải chọn u_1 sao cho tổng trên đạt giá trị cực đại.

Từ những điều trên và theo định nghĩa ta viết được:

$$f_T(x_1) = \max_{0 \le u_1 \le x_1} \{g(u_1) + h(x_1 - u_1) + f_{T-1}[au_1 + b(x_1 - u_1)]\} \tag{9.5}$$

Phương trình (9.5) gọi là phương trình hàm Bellman. Giải bài toán bằng quy hoạch động là giải hệ phương trình (9.5), (9.4).

Phương trình hàm (9.5) đã thể hiện được bản chất của nguyên lí tối ưu là: nếu cả quá trình đã nhận phương án tối ưu thì không phụ thuộc vào x_1 và u_1, trên quá trình T – 1 năm sau lời giải cũng phải làm thành phương án tối ưu ứng với trạng thái ban đầu $x_2 = au_1 + b(x_1 - u_1)$ là kết quả tác động của điều khiển ban đầu u_1.

Nếu ở năm đầu ta chọn U_1^* tối ưu cho cả quá trình thì ứng với nó ta cũng có $f_{T-1}(a u_1^* + b(x_1 - u_1^*))$ là lợi nhuận tối đa trên T-1 năm sau cùng với việc sử dụng phương án tối ưu cho cả quá trình I năm. Từ đó có hệ quả quan trọng sau đây:

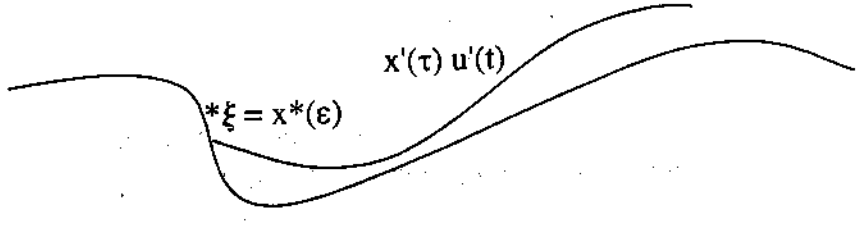
(*) Nếu cả quá trình nhận phương án tối ưu thì trên từng đoạn cũng phải là tối ưu.

Ngược lại ta không thể nói riêng trên mỗi đoạn tối ưu thì cả quá trình tối ưu. Ở đây ta có thể nói thêm rằng nếu tại một thời điểm nào đấy của quá trình điều khiển được chọn không phải là tối ưu thì việc xét tiếp phương án đó trên những bước sau là vô ích, vì dù trên những bước sau cố gắng đạt tối ưu thì cả phương án trong cả quá trình là không thể tối ưu.

9.2.5.2. Bài toán liên tục

Ở đây ta sẽ xét phương trình hàm của bài toán liên tục, bài toán điều khiển tối ưu được nêu trong phần 9.2.1 chương 9.

Giả sử $x^*(t)$, $t \in [0, I]$ là quỹ đạo tối ưu ứng với điều khiển tối ưu $U^*(t)$ bài toán (1), (2). Lấy một điểm $\xi = x^*(\varepsilon)$ trên quỹ đạo tối ưu, theo nguyên lý tối ưu:



thì đoạn từ $\xi = x^*(\varepsilon)$ tới $x^*(T)$ cũng phải là tối ưu theo nghĩa cho phiến hàm sau dãy giá trị cực tiểu:

$$I\varepsilon(\xi, u') = \int_0^I (x,u)d0 \qquad (9.6)$$

Tức là nếu xếp phiến hàm $I_\tau(\Im, u)$ trên tất cả các quỹ đạo cho phép của phương trình (9.1) ứng với các điều khiển cho phép $u(t)$, $t \in [\tau, T]$ thì $I_\tau(\Im, u)$ sẽ đạt giá trị cực tiểu trên quỹ đạo $x^*(t)$, $t \in [\tau, T]$ ứng với điều khiển $u^*(t)$ giống như cách định nghĩa lợi nhuận tối đa, ở đây ta đặt:

$$g(\Im,c) = \min_{\substack{u(0)\in u \\ 0\in[\tau,T]^{\tau}}} \int^{T} f^{0}(x(0),u(0))d0 \qquad (9.7)$$

Rõ ràng với $\tau = 0, \Im = x_0$ ta có $g(x_0, 0)$ là giá trị cực tiểu của phiến hàm I(u) trong công thức (9.2).

Trong bài toán liên tục ta có hai giả thiết quan trọng sau:

Giả thiết 1: Từ bất kỳ một điểm $\Im = x(\tau), \tau \in [0,T]$ trong không gian trạng thái, khác điểm x(t), bao giờ cũng tồn tại một quỹ đạo chuyển động tới x(T) giả thiết này đảm bảo cho hàm $g(\Im, c)$ được xác định trong không gian trạng thái. Hay nói cách khác để cho mọi điểm x(t) bao giờ cũng tồn tại một điểm $U_x(t)$ đưa hệ từ x(t) tới x(T) và cho phiến hàm trong (9.6) giá trị cực tiểu. Do đó, ta có thể xét hàm g(x, t), $x \in E^n, 0 \le t \le T$; Nhớ lại rằng x là véctơ n - chiều: $x = (x^1, x^2, ..., x^n)$.

Vì g(x, t) là một hàm của t và n biến X^y nên ta có thể nối tính liên tục và đạo hàm của nó.

Giả thiết 2:

Hàm g(x, t) liên tục khắp nơi và có đạo hàm liên tục theo x^i, t:

$$\frac{\partial g}{\partial x^1}, \frac{\partial g}{\partial x^2}, ..., \frac{\partial g}{\partial x^n}, \frac{\partial g}{\partial t}$$

Hàm số g(x,t), với những giả thiết trên ta sẽ thoả mãn phương trình hàm sau đây, gọi là hàm phương trình BELLMAN.

$$\frac{-\partial g}{\partial t} = \min_{u\in U}\left[f^{0}(x,u) + \sum_{i=0}^{n} f^{i}(x,u)\frac{\partial g}{\partial x^{i}} \right] \qquad (9.8)$$

Với ràng buộc:

$$g(x,T) = 0 \qquad (9.9)$$

Điều kiện (9.9) suy ra từ (9.7) nếu đặt $\tau = T$.

Sử dụng ký hiệu grad $g(x, t) = \left\{ \frac{\partial g}{\partial x'}, ..., \frac{\partial g}{\partial x''}, \frac{\partial g}{\partial t} \right\}$

Phương trình (9.8) sẽ được viết dưới dạng:

$$-\frac{\partial g}{\partial t} = \min_{u \in U}\left[f^0(x, u) + (\text{gradg}, f(x, u)) \right] \qquad (9.10)$$

Ở đây phương trình (9.10) là phương trình vi phân đạo hàm riêng bậc 1, phi tuyến.

Vì điều khiển u cho phiến hàm trong công thức (9.8) giá trị cực tiểu sẽ có giá trị ứng với từng cặp giá trị x và giá trị gradg, nên u là một hàm của x và gradg:

$$u = u(x, \text{gradg}) \qquad (9.11)$$

Phương pháp vừa nêu trên đây có thể giải quyết trong trường hợp thoả mãn các giả thiết 1, 2. Trong thực tế thì hàm số g(x,t) nói chung không khả vi khắp nơi nên phương pháp trên gặp phải khó khăn lớn.

9.2.6. Sơ đồ chung giải bài toán quy hoạch động

Như đã nói ở phần trên, để giải bài toán bằng quy hoạch động trên máy tính điện tử người ta thường rời rạc hoá, thường chia không gian - thời gian thành mạng lưới. Và cũng chú ý rằng nhờ có máy tính điện tử thì quy hoạch động mới được áp dụng rộng rãi.

Trước khi nêu sơ đồ chung giải bài toán bằng quy hoạch động ta cần chú ý mấy nét đặc biệt sau đây:

1. Muốn dùng sơ đồ chung của quy hoạch động thì bài toán phải đưa về dạng nhiều bước nối tiếp nhau.

2. Hàm mục tiêu có dạng hàm cộng được.

3. Quy nạp ngược. Bài toán quy hoạch động phải thoả mãn giả thiết tồn tại nghiệm tối ưu. Phép quy nạp ngược thể hiện ở chỗ là nếu thứ tự thời gian tăng từ 0, 1, 2, ..., T. Thì thứ tự bước sẽ bắt đầu từ T, T-1, ..., 0. song không nhất thiết khi nào cũng phải sử dụng việc xét bước trong thứ tự ngược.

Trong sơ đồ giải tổng quát ta xét các phương trình hàm sau đây:

$$f_1(x) = \max_{0 \le u \le x} \{g(u) + h(x - u)\} \qquad (9.12)$$

$$f_T(x_1) = \max_{0 \le u_1 \le x_1} \{g(u_1) + h(x - u_1) + f_{T-1}[au_1 + b(x_1 - u_1)]\} \qquad (9.13)$$

Phương trình (9.12) là ứng với quá trình một bước, phương trình (9.13) là ứng với T bước.

9.2.6.1. Quá trình 1 bước: Ở đây ta chỉ việc chọn u sao cho biểu thức trong dấu móc (9.12) đạt giá trị cực đại, với ràng buộc: $0 \le u \le x$

9.2.6.2. Quá trình 2 bước

Từ phương trình (9.13) ta có:

$$f_2(x_1) = \max_{0 \le u_1 \le x_1} \{g(u_1) + h(x_1 - u_1) + f_1[au_1 + b(x_1 - u_1)]\}$$

$$= \max_{0 \le u \le x} \left\{ g(u_1) + h(x_1 - u_1) + \max_{0 \le u_2 \le x_2} [g(u_2) + h(x_2 - u_2)] \right\}$$

Ở đây: $x_2 = au_1 + b(x_1 - u_1)$

hay: $x_2 = x_2(u_1)$

Có nghĩa là ứng với mỗi giá trị $x_2 = au_1 + b(x_1 - u_1)$ (trạng thái thu được do tác động của u_1 tương ứng) ta sẽ chọn được một giá trị u_2^* giữa những điều khiển u_2 nằm trong khoảng $0 \le u_2 \le x_2$, sao cho lợi nhuận trên năm thứ hai là cao nhất.

$$f_1(x_2) = \max_{0 \le u_2 \le x_2} \{g(u_2) + h(x_2 - u_2)\} = g(u_2^*) + h(x_2 - u_2^*)$$

và ta có thêm: $u_2^* = u_2^*(x_2)$ hay $u_2^*(u_1)$ (*)

Để xét quá trình của cả hai năm, thì giữa tất cả những điều khiển u_1 thuộc $\{0, x_1\}$ ta phải chọn vì sao lợi nhuận của cả hai là tối đa, nghĩa là những giá trị $f_1(au_1^* + b(x_1 - u_1^*))$ đã được tính phụ thuộc vào u_1^* trên bước thứ nhất, và bởi vì các hàm g, h cho nên việc lấy cực đại trên bước thứ hai hoàn toàn xác định. Hơn nữa điều khiển u_1 biến thiên trong khoảng xác định $\{0_1, x_1\}$ nên ở bước cuối cùng này giá trị u_1 hoàn toàn xác định. Biết được u_1^* cho giá trị $f_1(x_1)$ thay vào (*) ta tìm được u_2^* và như vậy (u_1^*, u_2^*) chính là phương án tối ưu cho cả quá trình của hai bước.

Ví dụ: Ta xét một quá trình hai năm chia làm hai bước với các số liệu ban đầu như sau:

$T = 2$	$a = 0,9$	$g(x) = 1,5$
$x = 1000^d$	$b = 0,8$	$h(x) = 2x$

Ta viết lại phương trình hàm:

$$f_2(1000) = \max_{0\le u_1\le x_1} = \{1,5u_1 + 2(1000-u_1) + f_1(0,9u_1 + 0,8(1000-u_1))\}$$

$$f_1(x_2) = \max_{0\le u_2\le x_2}\{g(u_2) + h(x_2-u_2)\} = \max_{u_2}\{1,5u_2 + 2(x_2-u_2)\}$$

$$= \max_{u_2}\{2x_2 - 0,5u_2\} = 2x_2;\ \text{với điều kiện } u_2^* = 0$$

Như vậy cho mọi x_2 lợi nhuận tối đa thu được trên năm thứ hai là $2x_2$ và điều khiển tối ưu cần phải lấy $u_2^* = 0$ thay giá trị vừa tính được vào phương trình đầu để xét ta có (chú ý: $x_2 = [au_1 + b(x_1 - u_1)]$).

$$f_2(1000) = \max_{0\le u_1\le x_1}\{1,5u_1 + 2(1000-u_1) + 2[0,9u_1 + 0,8(1000-u_1)]\}$$

$$= \max\{3600 - 0,3u_1\} = 3600;$$

$u_1^* = 0$

Như vậy phương án tối ưu là $u_1^* = 0$, $u_2^* = 0$ và lợi nhuận tối đa là:

$$f_2(1000) = 3600$$

Ta thu được kết quả: với số vốn ban đầu là $1000^đ$ dùng mua thiết bị sản xuất hai mặt hàng A, B với số lãi như trên thì tốt nhất là dùng mua thiết bị để sản xuất mặt hàng B trong cả hai năm. Trước khi xét trường hợp tổng quát, ta xét thêm một quá trình 3 bước, 3 năm.

9.2.6.3. Quá trình 3 bước

$$f_3(x_1) = \max_{u_1\in[0,x_1]}\{g(u_1) + h(x_1-u_1) + f_2[au_1 + b(x_1-u_1)]\}$$

Ta bắt đầu xét trên năm cuối cùng:

- *Bước 1*: Ứng với mỗi trạng thái $x_3 = au_2 + b(x_2 - u_2)$ là kết quả tác động của điều khiển u_2, ta sẽ chọn được điều khiển $u_3^* \in [0,x_3]$ sao cho lợi nhuận trên năm thứ 3 tối đa là:

$$f_1(x_3) = \max_{0\le u_3\le x_3}\{g(u_3) + h(x_3-u_3)\}$$

$$= g(u_3^*) + h(x_3 - u_3^*)$$

Đồng thời ta thu được: $u_3^* = u_3^*(x_3)$

Vì $x_3 = au_2 + b(x_2 - u_2)$ nên x_3 là một hàm của u_2, và cũng như vậy, ta có:

$$u_3^* = u_3^*(u_2)$$

- *Bước 2*: Ứng với mỗi trạng thái $x_2 = au_1 + b(x_1 - u_1)$, vốn ban đầu cho cả hai năm sau cùng (là trạng thái kết quả tác động của điều khiển u_1) ta sẽ chọn được một điều khiển u_2^* trong khoảng $[0, x_2]$ sao cho lợi nhuận thu được trên cả năm sau cùng là tối đa:

$$f_2(x_2) = \max_{0 \le u_2 \le x_2} \{g(u_2) + h(x_2 - u_2) + f_1[au_2 + b(x_2 - u_2)]\}$$

$$= g(u_2^*) + h(x_2 - u_2^*) + f_1(au_2^* + b(x_2 - u_2^*))$$

Đồng thời ta thu được mối phụ thuộc:

$$u_2^* = u_2^*(u_1)$$

Bước 3: Trên bước cuối cùng, để tìm u_1 sao cho lợi nhuận thu được trên cả 3 năm là tối đa thì chỉ việc lấy cực đại:

$$f_3(x_1) = \max_{0 \le u_1 \le x_1} \{g(u_1) + h(x1 - u_1) + f_2[au_1 + b(x_1 - u_1^*)]\}$$

$$= g(u_1^*) + h(x_1 - u_1^*) + f_2[au_1^* + b(x_1 - u_1)]$$

Vì ở trên bước cuối cùng x_1 là một giá trị cụ thể nên u_1^* cũng có giá trị cụ thể. Biết được $u_2^* = u_2^*(u_1^*)$, $u_3^* = u_3^*(u_2^*)$ ta sẽ tìm được phương án tối ưu:

$$(u_1^*, u_2^*, u_3^*)$$

cho lợi nhuận tối đa của quá trình 3 năm:

$$f_3(x_1) = g(u_1^*) + h(x_1 - u_1^*) + f_2[au_1^* + b(x_1 - u_1^*)]$$

Để làm thí dụ cho trường hợp này, bạn đọc có thể thay T = 3 vào ví dụ vừa xét ở trên đây và giải tìm giá trị của lợi nhuận tối đa và phương án tối ưu tương ứng cho quá trình 3 năm, 3 bước.

9.2.6.4. Quá trình T năm, T bước

Phương trình hàm tổng quát là:

$$f_T(x_1) = \max_{0 \le u_1 \le x_1} \{g(u_1) + h(x_1 - u_1) + f_{T-1}[au_1 + b(x_1 - u_1)]\}$$

1. Ta bắt đầu xét quá trình từ năm cuối T. Ứng với mỗi trạng thái:

$$x_T = au_{T-1}(au_1 + b(x_{T-1} - u_{T-1})$$

(vốn ban đầu cho năm thứ T là kết quả tác động của điều khiển u_{T-1}) ta sẽ chọn một điều khiển u_T^* trong khoảng $[0,x_{T-1}]$ sao cho lợi nhuận trên năm cuối cùng là tối đa:

$$f_1(x_T) = \max_{0\le u_T \le x_T} [g(u_T) + h(x_T - u_T)] = g(u_T^*) + h(x_T - u_T^*)$$

và $\quad u_T^* = u_T^*(u_{T-1})$ (a.1)

2. Ứng với mỗi điều khiển ban đầu u_{T-2} và trạng thái ban đầu $x_{T-1} = au_{T-2} + b(x_{T-2} - u_{T-2})$ (vốn đầu tư cho 2 năm cuối, là kết quả tác động của u_{T-2}) ta sẽ chọn được một điều khiển u_{T-1}^* trong khoảng $[0,x_{T-1}]$ sao cho lợi nhuận của cả hai năm cuối cùng là tối đa:

$$f_2(x_{T-1}) = \max_{0\le u_{T-1}\le x_{T-1}} \{g(u_{T-1}) + h(x_{T-1} - u_{T-1}) + f_1[au_{T-1} + b(x_{T-1} - u_{T-1})]\}$$

$$= g(u_{T-1}^*) + h(x_{T-1} - u_{T-1}^*) + f_1\left[au_{T-1}^* + b(x_{T-1} - u_{T-1}^*)\right]$$

Và ta có: $\quad u^*_{T-1} = u^*_{T-1}(u_{T-2})$ (b.1)

3. Trên bước thứ k: Ứng với mỗi điều khiển ban đầu u_{T-k} và trạng thái ban đầu $x_{T-k+1} = au_{T-k} + b(x_{T-k} - u_{T-k})$ (vốn đầu tư cho k năm cuối, là kết quả tác động của điều khiển u_{T-k}) ta sẽ chọn được một u_{T-k+1}^* trong khoảng $[0, x_{p-k+1}]$ sao cho tổng lợi nhuận thu được trên k năm cuối là tối đa:

$$f_k(x_{T-k+1}) = \max_{0\le u_{T-k+1}\le x_{T-k+1}} \left\{\begin{matrix} g(u_{T-k+1}) + h(x_{T-k+1} - u_{T-k+1}) + \\ f_{k-1}\left[au_{k-T+1} + b(x_{T-k+1} - u_{T-k+1})\right] \end{matrix}\right\}$$

$$= g(u_{T-k+1}^*) + h(x_{T-k+1} - u_{T-k+1}^*) + f_{k-1}\left[au_{T-k+1}^* + b(x_{T-k+1} - u_{T-k+1}^*)\right]$$

đồng thời ta thu được mối phụ thuộc:

$$u_{T-k+1}^* = u_{T-k+1}^*(u_{T-k}) \quad \text{(c.1)}$$

4. Trên bước thứ T – 1 (tức là trên năm thứ hai): Ứng với mỗi điều khiển u_1 trên năm đầu và trạng thái ban đầu và $x_2 = au_1 + b(x_1 - u_1)$ (vốn đầu tư cho T – 1 năm sau, kết quả do tác động của điều khiển ban đầu u_1), ta chọn được một điều khiển:

$u_2 \in [0, x_2]$ sao cho tổng lợi nhuận trên T – 1 năm sau là tối đa:

$$f_{T-1}(x_2) = \max_{u_2 \in [0, x_2]} \left\{ g(u_2) + h(x_2 - u_2) + f_{T-2}\left[au_2 + b(x_2 - u_2)\right] \right\}$$

$$= g(u_2^*) + h(x_2 - u_2^*) + f_{T-2}\left[au_2^* + b(x_2 - u_2^*)\right]$$

và $$u_2^* = u_2^*(u_1). \qquad (d.1)$$

5. Trên bước cuối cùng ta trở về đầu quá trình. Ở đây việc chọn u_1 trong $[0, x_1]$ để cho tổng lợi nhuận trên năm đầu và T – 1 năm sau là tối đa sẽ gói gọn trong phép lấy cực đại:

$$f_T(x_1) = \max_{0 \le u_1 \le x_1} \left\{ g(u_1) + h(x_1 - u_1) + f_{T-1}\left[au_1 + b(x_1 - u_1)\right] \right\}$$

Vì x_1 là vốn ban đầu đã cho nên giá trị u_1^* hoàn toàn xác định. Như vậy, đến đây ta đã tính được giá trị tối ưu của hàm mục tiêu $f_T(x_1)$ của quá trình T năm với vốn ban đầu là x_1:

$$f_T(x_1) = g(u_1^*) + h(x_1 - u_1^*) + f_{T-1}\left[au_1^* + b(x_1 - u_1^*)\right]$$

Việc xác định phương án tối ưu tương ứng chỉ còn phải thay u_1^* vào (d.1) tìm u_2^*, u_3^*, lần lượt vào (c.1), (b.1), (a.1) để tìm các điều khiển tiếp theo.

9.2.7. Nhận xét và kết luận

Trong sơ đồ giải chúng ta thấy từ một bài toán tìm nhiều biến đã được đưa về dạng nhiều bài toán một biến.

Trên mỗi bước trung gian lời giải mang tính chất quy ước nghĩa là điều khiển u_t được chọn tại thời điểm t, tối ưu đối với trạng thái x(t) mà thôi.

Số bước của quá trình là tuỳ thuộc vào từng bài toán: có bài toán số bước đã cố định, có bài toán số bước chưa biết trước. Có loại bài toán lại cần phải xác định số bước bao nhiêu thì tối ưu.

Nói chung dùng quy hoạch động rất ưu việt trong các quá trình rời rạc. Số bài toán cho phép thu được lời giải tích trực tiếp từ phương trình hàm Bellman không nhiều. Những bài toán này cho phép ta xây dựng phương án tối ưu một cách chính xác. Những bài toán không thu được lời giải giải tích thì ta dùng phương pháp gần đúng. Cụ thể là chia miền biến thiên của

các tham số và xét các điểm nút. Ở đây sẽ liên quan tới bài toán nhiều chiều. Số chiều của một bài toán được xác định bằng số lượng các tham số điều khiển và các tham số pha. Nếu số lượng tham số ít và số điểm chia vừa phải thì số lượng phép tính không nhiều lắm, nhưng nếu số điểm chia tăng lên (độ chính xác cao hơn) thì số lượng phép tính tăng lên rất nhanh. Ta có thể thấy ngay là nếu trong miền ta xét mỗi biến nhận 10 giá trị, đối với bài toán 2 chiều phải xét 100 nút, ba chiều - 1000 nút.

So với phương pháp tổ hợp thì quy hoạch động đã giảm bớt số lượng tính toán đi rất nhiều. Xét một quá trình N bước và trên mỗi bước lời giải có k khả năng. Như vậy nếu dùng phương pháp tổ hợp thì phải xét tất cả là k^N khả năng.

Dùng quy hoạch động đưa về xét lời giải trên từng bước. Mỗi bước phải xét k phương án, toàn bộ N bước là N × k phương án.

Tỷ số $\frac{N \times k}{k^N} = N \times k^{-N+1}$ chỉ rõ mối quan hệ giữa hai phương pháp.

Lấy một ví dụ cụ thể: N = 10, k = 5 thì số tổ hợp là 5^{10} tức là hơn $9{,}76.10^6$. Trong khi đó dùng quy hoạch động thì chỉ cần phân tích 50 phương án là được. Ngoài việc giảm bớt số lượng tính toán, quy hoạch động còn rất ưu việt trong việc xét các loại ràng buộc. Trong quy hoạch toán học, khó khăn thường gặp phải là các loại ràng buộc phi tuyến, các hàm mục tiêu phi tuyến. Nhưng đối với quy hoạch động, cái đó không gây khó khăn gì, ngược lại nó còn thu nhỏ, rút bớt miền cần xét, giảm bớt được số lượng cần tính toán.

9.3. ÁP DỤNG QUY HOẠCH ĐỘNG TRONG THIẾT KẾ XÂY DỰNG ĐƯỜNG

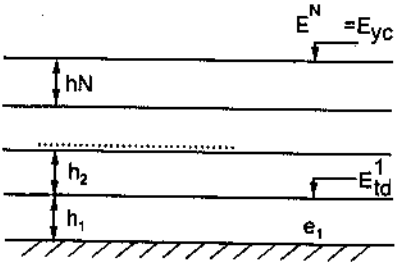
9.3.1. Thiết lập bài toán

Trong thiết kế đường, ta có bài toán:

Cần thiết kế mặt đường mềm có n lớp (hình 9.2) mỗi lớp là một loại vật liệu với cường độ là e_i, độ dày là h_i, giá quy đổi là g_i. Trong đó h_i, g_i có thể là các hàm phi tuyến phức tạp, được tính toán theo các nguyên lý thiết kế đường, dựa trên các kết quả nghiên cứu thực nghiệm. Kết cấu mặt đường được thiết kế dựa trên nguyên lý tính tải trọng tĩnh tăng dần hoặc

biến dạng tích luỹ sinh ra cho mặt đường chịu tải trọng trùng phục nhiều lần không chịu vượt quá trị số biến dạng cho phép.

Như vậy, ở đây ta lấy biến dạng thẳng đứng là chỉ tiêu đặc trưng kĩ thuật khi thiết kế mặt đường. Công nhận nguyên lí này, dựa vào công thức tính ứng suất trong hệ bán không gian đàn hồi của Butsinet và theo đề nghị đổi tầng của Pokrovski. G. I người ta đưa ra công thức tính module biến dạng tương đương cho hệ hai tầng như sau:



Hình 9.2

$$E_{td}^{1} = \frac{E_0}{1 - \frac{2}{\pi}\left(1 - \frac{1}{w^{3,5}}\right)\text{arctg}\frac{nh}{D}} \qquad (9.14)$$

trong đó, E_0 là cường độ nền đất, D là đường kính của mặt ép cứng tròn, h là bề dày của lớp vật liệu:

$$w = (e_1 / E_0)^{1/2,5}$$

trong đó e_1 là cường độ vật liệu. Từ công thức (9.14) qua một số phép biến đổi ta rút ra được:

$$h = \frac{D}{w}\text{tg}\left[\frac{1 - E_0 / E_{td}^{1}}{\frac{2}{\pi}(1 - 1/w_i^{3,5})}\right] \qquad (9.15)$$

Trên thực tế những đoạn đường cao cấp thường được thiết kế theo nhiều tầng, người ta thường làm như sau:

Nếu ta bắt đầu tính từ dưới cùng lên thì E_0, h_1 và e_1 coi như đã biết. Từ công thức (9.14) ta suy ra E_{td}^{1}. Sau đó, giả thiết E_{td}^{1} đóng vai trò E_0 và với những giá trị e_2, h_2 ta lại tính ra E_{td}^{1}. Bởi vì chỉ có một phương trình (9.14) (hoặc (9.15)) nên trong bất kì trường hợp nào cũng phải cho biết 3 trong 4 tham số: bề dày h, cường độ vật liệu e_1, cường độ E_0 và E_{td}^{1}. Nếu tầng trên cùng E_{td} bị giới hạn bằng E_{yc} thì ta có thể coi ẩn số là h. Nếu

cho biết h (thường lớp trên cùng vật liệu rất đắt nên dùng h là bề dày tối thiểu cho phép) thì ẩn số sẽ là một trong ba tham số kia.

Tóm lại theo cách này hoặc làm từ dưới lên, hoặc làm từ trên xuống, hoặc làm từ hai phía vào giữa vẫn phải cho biết 3 trong 4 tham số và giải phương trình (9.14) hoặc (9.15) đối với tham số còn lại. Và trong thực tế thì chỉ thử được một vài phương án, sau đó theo kinh nghiệm người thiết kế sẽ chọn ra phương án sử dụng. Vì thế, dùng cách tính như vậy chưa cho ta lời giải tối ưu cho cả quá trình.

Áp dụng quy hoạch toán học ta có thể cho phương pháp tính toán các đường theo (9.14) đã nói ở trên một thuật toán tối ưu kết cấu cho lớp mặt đường. Vấn đề được đặt ra như sau:

Giả sử phải rải N lớp vật liệu làm đường đối với các cường độ e_i đã biết, cho biết cường độ nền đất E_0 và E_{yc}. Hỏi phải rải mỗi lớp dày bao nhiêu để đảm bảo điều kiện kĩ thuật và cho giá thành xây dựng nhỏ nhất, giá thành được biểu diễn bằng hàm cộng được:

$$G(h_1, h_2, ..., h_N) = \sum_{i=1}^{N} g_i(h_i) \qquad (9.16)$$

Việc đưa giá thành về hàm số $g_i(h_i)$ là cơ sở, vì ta giả thiết rằng tại nơi xây dựng đơn giá vật liệu và thi công đã biết. Như vậy giá thành mỗi lớp chỉ phụ thuộc vào bề dày của nó mà thôi. Chú ý rằng $g_i(h_i)$ có thể là một hàm tuyến tính hay là phi tuyến tính của đối số h_i. Dưới đây sẽ xác lập mô hình toán học của bài toán này.

9.3.2. Mô hình toán học

Sử dụng công thức tính bề dày và cường độ vật liệu (9.14) (9.15) ta sẽ đưa bài toán về dạng sau:

Đối với hệ hai tầng (nền đất và một lớp vật liệu)

$$h_i = \frac{D}{w_1} tg \left\{ \frac{[1 - E_0 / E_1]}{\left[\frac{2}{\pi}(1 - 1/w_1^{3,5})\right]} \right\} \qquad (9.17)$$

Theo cách tính đổi tầng đối với hệ nhiều tầng, trong bài toán N tầng ta có:

$$h_i = \frac{D}{\underset{i=1,2,...,N}{n_i}} tg \left\{ \frac{\left[1 - E_i - 1/E_i\right]}{\left[\frac{2}{\pi}(1 - 1/w_i^{3,5})\right]} \right\} \quad (9.18)$$

Trong đó: E_0 là cường độ nền đất;

E_i là các E_{td};

E_n là E_{yc};

$w_i = \sqrt[2,5]{\frac{e_i}{E_{i-1}}}$, e_i là cường độ vật liệu thứ i.

Như vậy ta phải khảo sát mô hình:

$$\min_{h_1, h_2, ..., h_n} \left\{ G(h_1, ..., h_N) = \sum_{i=1}^{N} g_i(h_i) \right\} \quad (9.19)$$

$$\left.\begin{aligned} E_0 &= l_0 \\ E_N &= E_{yc} \end{aligned}\right\} \quad (9.20)$$

$$h_m \le h_i \le h_M \quad (9.21)$$

Đây là một bài toán điều khiển tối ưu và được phát biểu như sau: hãy chọn các tham số điều khiển h_i để cho hệ chuyển động từ trạng thái ban đầu $E_0 = e_0$ tới trạng thái cuối cùng $E_n = E_{yc}$ một cách tối ưu theo nghĩa cho hàm mục tiêu (9.16) giá trị nhỏ nhất. Ở đây các tham số E_i là các tham số trạng thái. Để giải bài toán trên ta sẽ biến đổi chút ít, từ (9.18) ta đặt:

$$h_i = f(E_{i-1}, E_i)$$

Khi đó:

$$G(h_1, ..., h_N) = \sum_{i=1}^{N} g_i(f(E_{i-1}, E_i)) = \sum_{i=1}^{N} G_i(E_{i-1}, E_i)$$

Ngoài những ràng buộc (9.20), (9.21) từ thực tế kĩ thuật ta có thêm:

$$\begin{cases} E_{i-1} \le E_i \le e_i \\ E_n - 1 \le E_{yc} \quad \text{nhận giá trị nguyên} \\ f(E_{i-1}, E_i) \end{cases}$$

Cuối cùng ta có :

$$\min_{E_1,\dots E_{N-1}} \left\{ G(E_0, E_1, \dots E_n) = \sum_{i=1}^{N} G_i(E_{i-1}, E_i) \right\} \tag{9.22}$$

$$\left.\begin{aligned} E_0 &= e_0 \\ E_n &= E_{yc} \end{aligned}\right\} \tag{9.23}$$

$$\left.\begin{aligned} E_{i-1} \le E_i &\le l_i \\ E_{n-1} &\le E_{yc} \end{aligned}\right\} \tag{9.24}$$

$$f(E_{i-1}, E_i) = 0, 1, 2, \dots, k$$

Ta đã đưa bài toán tìm điều khiển tối ưu về xét trong không gian trạng thái E_i. Giải bài toán (9.22), (9.23), (9.24) sẽ cho ta véctơ trạng thái tối ưu $E^* = (E_0, E_1^*, \dots, E_{N-1}^*, E_N)$ và tương ứng với nó là véctơ điều khiển tối ưu:

$$h^* = (h_1^*, \dots, h_N^*)$$

9.3.3. Thuật toán

Trong bài toán trên các hàm $G_i (E_{i-1}, E_i)$ là phi tuyến nên không thể dùng quy hoạch tuyến tính được. Hơn nữa nó có dạng tg(x) nên nếu khoảng xác định lớn hơn π thì cũng không phải là hàm lồi lõm, sẽ có điểm đặc biệt giá trị của hàm bằng vô cùng, vì vậy cũng không áp dụng trực tiếp quy hoạch phi tuyến được. Ta nhận thấy ở đây hàm mục tiêu là một hàm cộng được, quá trình chia thành từng bước nên ta có thể sử dụng quy hoạch động.

Nếu ta gọi chi phí cho i lớp theo phương án tối ưu là

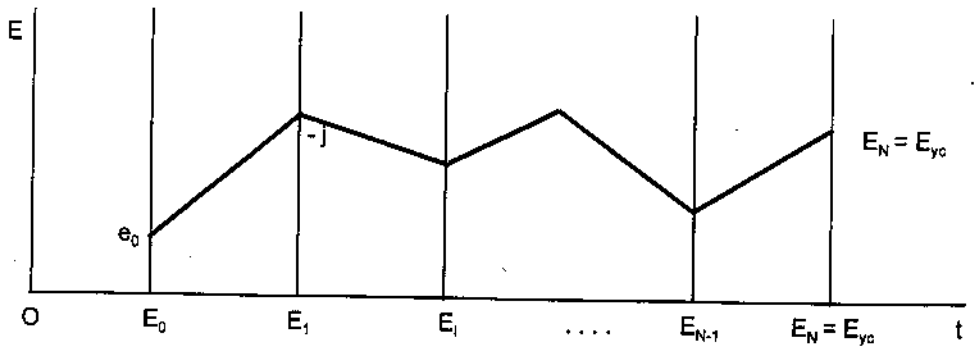
$$R_i(E_0, E_1, \dots, E_N) = \max_{0 \le E_1 \le l_1} \{G_1(E_0, E_1) + R_{N-1}(E_1, \dots, E_N)\} \tag{9.25}$$

Phương trình (9.25) sẽ được giải theo bằng một thuật toán sau đây, theo phương pháp gần đúng:

- Trên mặt phẳng dựng trục OE, OT.

Trên OT ta dựng các nửa đường thẳng song song với OE cách đều nhau, chúng sẽ là miền biến thiên của các E_i (xem hình vẽ). Nếu trên mỗi nửa đường thẳng đó ta lấy một điểm E_iJ_i thì ta được một véctơ trạng thái:

$$E = (E_{0J_0}, E_{1J_1}, ..., E_{N-1J_{N-1}}, E_N)$$



Hình 9.3

ứng với nó hàm G (E_0, ..., E_N) có một giá trị xác định nếu ta nối các điểm $E_{i_{J_i}}$ với $E_{i+1_{J_{i+1}}}$ thì sẽ nhận được một đường gấp khúc từ E_0 đến E_N. Một cách rất trực giác ta có thể nói rằng bài toán trở về chọn một con đường gấp khúc từ E_0 đến E_N sao cho độ dài tương ứng (giá trị hàm số G(E_0, ..., E_N)) là nhỏ nhất.

Chú ý rằng trên mỗi nửa đường thẳng E_i, giá trị E_i có thể nhận liên tục, nhưng để tính toán trên MTĐT ta chỉ lấy một số điểm hữu hạn, cụ thể trên mỗi E_i ta chỉ lấy q_i điểm. Thêm nữa, khoảng cách giữa các đường song song E_i không giữ một vai trò gì trong thuật toán nên để cho tiện ta lấy chúng bằng nhau: $\lambda_i = \lambda$ để cho mọi i.

Thứ tự các bước như sau:

Bước 1:

Trên $t = 0$ xác định $E_0 = e_0$

Để cho mỗi điểm $E_{i_{J_i}} \in \{t = \Im\}, J_i \in \{1, 2, 3..., q_i\} = Q_i$

Ta biểu diễn độ dài của đoạn thẳng nối nó với E_0 theo phương án tối ưu là: $l(E_{i_{J_i}}) = G(E_0, E_{i_{J_i}})$

Vì để cho mỗi $E_{i_{J_i}}$ chỉ có một đường nối nó tới E_0 nên chính nó là đường tối ưu đối với $E_{i_{J_i}}$

Bước 2:

Sử dụng kí hiệu $\forall a$, đọc là để cho mọi giá trị của a, ta có:

$$\forall E_{2_{J_1}} \in \{t = 2\Im\}, J_2 \in \{1, 2, 3..., q_2\} = Q_2$$

$$l(E_{2_{J_2}}) = \min_{J_1 \in Q_1} \left\{ l(E_{1J_1}) + G_2(E_{1_{J_1}}, E_{2_{J_2}}) \right\} = l(E^*_{1_{J_1}}) + G_2(E^*_{1_{J_1}}, E_{2_{J_2}})$$

và ta có mối phụ thuộc:

$$E^*_{1_{J_1}} = E^*_{1_{J_1}}(E^*_{2_{J_2}}) \tag{9.26}$$

Bước thứ i: $\forall E_{i_{J_1}} \in \{t = i\Im\}, J_i \in \{1, 2, ..., q\} = Q_i$

$$l(E_{iJ_i}) = \min_{J_{i-1} \in Q_{i-1}} \left\{ l(E_{i-1_{J-1}}) + G_i(E_{i-1_{J-1}}, E_{ij}) \right\}$$

$$= l(E^*_{i-1_{J-1}}) + G_i(E^*_{i-1_{J_{i-1}}}, E_{i_{J_i}})$$

và ta có mối phụ thuộc:

$$E^*_{i-1_{J_{i-1}}} = E^*_{i-1_{J_i}}(E_{i_{J_i}}) \tag{9.27}$$

Bước thứ N-1: $\forall E_{N\text{-}1\ J_{N-1}} \in \{t = (N\text{-}1)i\Im\}, J_{N-1} \ni \{1,2,...,q_{N-1}\} = Q_{N-1}$

$$l(E_{N-1_{J_{N-1}}}) = \min_{Jn-2 \in Qn-2} = \{l(E_{N\text{-}2\ j_{N-2}}) + G_{N-1}(E_{N\text{-}2\ J_{N-2}}, E_{N\text{-}1\ J_{N-1}})\}$$

$$= l(E^*_{N\text{-}2\ J_{N-2}}) + G_{N-1}(E^*_{N\text{-}2\ J_{N-2}}, E_{N\text{-}1\ J_{N-1}})$$

và ta có mối phụ thuộc:

$$E^*_{N\text{-}2\ j_{N-2}} = E^*_{N\text{-}2\ J_{N-2}}(E_{N\text{-}1\ j_{N-1}}) \tag{9.28}$$

Bước thứ N:

Vì $E_n = E_{yc}$ là một giá trị duy nhất nên:

$$l(E_N) = \min_{Jn-1 \in Qn-1} \left\{ l(E_{n-1_{J_{n-1}}}, E_n \right\}$$

$$= l\,(E^*_{N\text{-}1\ J_{N-1}}) + G_N(E^*_{N\text{-}1\ J_{N-1}}, E_N) \tag{9.29}$$

Và ta có mối phụ thuộc:

$$E^*_{N\text{-}1\ J_{N-1}} = E^*_{N\text{-}1\ J_{N-1}}(E_N) \tag{9.30}$$

Như vậy quá trình xác định giá trị tối ưu của hàm mục tiêu kết thúc trên bước N này và nó là một giá trị cụ thể (công thức (9.29)). Việc

xác định giá trị các E^*_{ij}; mà đường tối ưu đi qua sẽ được bắt đầu từ công thức (9.30). Từ (9.28), với giá trị cụ thể của E_N ta tìm được $E^*_{N-1}{}_{J_{N-1}}$. Tiếp tục như vậy theo (9.26) ta xác định các điểm tối ưu tiếp sau đó kết quả là ta thu được véctơ trạng thái tối ưu:

$$E^* = *(E_0, E_1^*, ..., E_{N-1}, E_N).$$

Và tương ứng với nó là véctơ điều khiển tối ưu

$$h^* = (h^*_1, ..., h^*_N)$$

Ở đây h^*_i sẽ được suy ra từ $h_i^* = f_i(E^*_{i-1}, E^*_i)$

Thuật toán trên đây cho phép ta tính với mọi số lớp N bất kỳ. Lời giải thu được mang tính chất gần đúng. Kết quả càng chính xác nếu số điểm chia các E_i dầy hơn. Nhưng như ta biết, số điểm chia K tăng lên thì số phép tính cũng tăng lên rất nhanh. Vì vậy trong khi tính toán nếu với cách chia K và K' mà kết quả không khác nhau mấy thì có thể dừng.

Sơ đồ chung giải bài toán quy hoạch động được áp dụng ở đây theo thứ tự cùng tăng của số bước và số lớp. Đối với bài này việc xét từ E_0 tới E_N hay ngược lại không quan trọng, từ cách làm và hình vẽ ta thấy kết quả sẽ như nhau trong cả hai quá trình.

9.3.4. Ví dụ

Để bạn đọc vận dụng được sơ đồ quy hoạch động, trong phần này sẽ trình bày thí dụ với những số liệu cụ thể.

Bài toán 1: Cần phải thiết kế kết cấu mặt đường 3 lớp với các loại vật liệu có cường độ tương ứng.

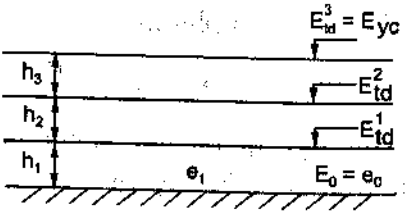
$l_1 = 300$, $l_2 = 625$, $l_3 = 1000$ và đơn giá $g_1 = 30$, $g_2 = 50$; $g_3 = 250$ đồng trên một đơn vị đo. Cường độ nền đất E_0 là 135, cường độ $E_{y0} = 390$.

Hãy xác định bề dày các lớp vật liệu đạt yêu cầu kĩ thuật cho tổng chi phí là bé nhất. Cho biết: $10 \le h_1 \le 30$; $8 \le h_2 \le 25$; $4 \le h_3 \le 20$. Theo cách tính từ trước tới nay vẫn dùng kết quả tính toán của kĩ sư (không xét vấn đề tối ưu) là:

$h_1 = 20$

$h_2 = 16$

$h_3 = 6$



và tính tổng chi phí là 2900 đồng.

Ta tính tay theo phương pháp quy hoạch động, ghi trong bảng sau:

E1	175	185	195	205	215	225	235	245	255	> 255
135	300 (10)	360 (12)	420 (14)	480 (16)	570 (19)	630 (21)	690 (23)	780 (26)	840 (28)	

E1 \ E2	285	295	305	315	325	335	345	355	365	375	385	> 385
175	600 (12)	650 (13)	700 (14)	750 (15)	800 (16)	850 (17)	900 (18)	950 (19)	1000 (20)	1000 (20)		
185	550 (11)	600 (12)	650 (13)	700 (14)	750 (15)	800 (16)	850 (17)	900 (18)	950 (19)	950 (19)	1000 (20)	
195		550 (11)	600 (12)	650 (13)	700 (14)	750 (15)	800 (16)	850 (17)	900 (18)	900 (18)	950 (19)	
205			550 (11)	600 (12)	650 (13)	700 (14)	750 (15)	800 (16)	850 (17)	850 (17)	850 (17)	
215				550 (11)	550 (11)	600 (12)	650 (13)	700 (14)	750 (15)	750 (15)	800 (16)	
225					550 (11)	550 (11)	600 (12)	650 (13)	700 (14)	750 (15)	800 (16)	
235						500 (10)	550 (11)	600 (12)	650 (13)	650 (13)	700 (14)	
245							500 (10)	550 (11)	600 (12)	650 (13)	700 (14)	
255							500 (10)	500 (10)	550 (11)	600 (12)	700 (14)	
390				1750 (7)	1750 (7)	1750 (7)	1500 (6)	1500 (6)	1250 (5)	1000 (4)	1000 (4)	

Công thức tính hàm giá thành suy từ công thức (9.16):

$$G_i(E_i - 1, E_i = g_i * \frac{D}{w_i} tg\left[\frac{1 - E_i / E_i - 1}{\frac{2}{\pi}\left(1 - 1 / w_i^{3,5}\right)}\right]$$

Kết quả trong các cột là: $\begin{Bmatrix} G_i \\ (H_i) \end{Bmatrix}$

Trong đó: G_i là giá thành tối ưu; H_i là bề dầy tương ứng.

Sử dụng chương trình tính trên máy tính điện tử, kết quả cho là:

$$h_1 = 10,\ h_2 = 18,\ h_3 = 4.$$

Và tổng chi phí là: 2200 đồng/1 đơn vị đo. Phần trăm lợi của phương pháp mới so với phương pháp cũ được tính như sau:

$$\frac{2900 - 2200}{2200} \approx 31{,}81\%$$

Theo công thức chung: $\dfrac{G_c - G_m}{G_m}$

Trong đó: G_c - Giá thành tính theo cách cũ; G_m - tính theo cách mới

Bài toán 2: Số liệu ban đầu:

$I_1 = 8000$	$G_1 = 30$	$8 \le h_1 \le 40$	$E_0 = 30$
$I_2 = 10000$	$G_2 = 40$	$8 \le h_2 \le 30$	$E_{yc} = 5700$
$I_3 = 20000$	$G_3 = 100$	$4 \le h_3 \le 15$	

Tính theo phương pháp cũ, lấy theo quy phạm:

$H_1 = 14, H_2 = 16, H_3 = 6$ và tổng chi phí là 3400 đồng/đơn vị đo.

Kết quả tính toán của máy tính là:

$h_1 = 18, h_2 = 8, h_3 = 4$ tổng chi phí là 1260 đồng/đơn vị đo.

Phần trăm lợi là 31,74%

Bài toán 3: Số liệu ban đầu:

$l_1 = 600$	$G_1 = 55$	$10 \le h_1 \le 30$	$E_0 = 100$
$l_2 = 867$	$G_2 = 50$	$8 \le h_1 \le 25$	$E_{yc} = 410$
$l_3 = 1000$	$G_3 = 250$	$3 \le h_1 \le 20$	

Kết quả tính theo phương pháp cũ của kĩ sư là:

$H_1 = 15, H_2 = 17, H_3 = 3$ và tổng chi phí là 2425 đồng/đơn vị đo.

Kết quả tính toán của máy tính là:

$H_1 = 10, H_2 = 17, H_3 = 3$ tổng chi phí là 2150 đồng/đơn vị đo.

Phần trăm lợi là 12,8%.

Theo sơ đồ Quy hoạch động, việc tính toán các loại bộ định hình 2, 3, 4 tầng đã cho kết quả rất khả quan. Theo kết quả thu được, giá trị % trung bình là xấp xỉ 32%.

Phụ lục

MỘT SỐ CHƯƠNG TRÌNH VÍ DỤ TỔNG HỢP

I. CHƯƠNG TRÌNH QUẢN LÝ TIỀN LƯƠNG - SỬ DỤNG CẤU TRÚC BẢN GHI (RECORD) VÀ FILE, MẢNG (ARRAY)

```
(* chuong trinh quan ly hoc luong can bo  SV thuc hien: Do gia ngoc*)
  uses crt,dos;
  type
     canbo=record
              macb:integer;
              ten:string[25];
              hsl:byte;
              pht,luong,Dl:longint;
              END;
  Var
 i:char;
procedure tao_file;
Var
f:file of canbo;
nguoi:canbo;
K:char;
x,y,i:integer;
tentep:string[25];
BEGIN clrscr;
     gotoxy(5,10);
     write('cho ten don vi');
     readln(tentep);
     assign(f,tentep);
```

```
rewrite(f);
K:='c';
nguoi.MaCB:=1;
While k='c' do
    BEGIN
       clrscr;
       gotoxy(20,5);write('PROGRAM NHAP SO LIEU');
       x:=12;y:=10;
       gotoxy(x,y);write('Ma CB:');
       gotoxy(x,y+1);write('TEN:');
       gotoxy(x,y+2);write('HSL:');
       gotoxy(x,y+3);write('PHT:');
       x:=x+9;
       With nguoi do
       BEGIN
          gotoxy(x,y);readln(MaCB);
          gotoxy(x,y+1);readln(Ten);
          gotoxy(x,y+2);readln(hsl);
          gotoxy(x,y+3);readln(pht);
       END;
       if nguoi.ten<>' ' then
         BEGIN
            write(f,nguoi);
            nguoi.MACB:=nguoi.MACB+1;
         END;
       gotoxy(5,20);
       write('Co tiep tuc nua khong?(C?K)');
       k:=readkey;
 END;
 write('Dang ghi file');
 close(f);
 END;
 {**********}
```

```
procedure Cap_nhat;
Var
f:file of canbo;
nguoi:canbo;
k:char;
x,y,n:integer;
st:string[80];
tentep:string[22];
i,error:integer;
r:longint;
BEGIN
  ClrScr;
gotoxy(5,10);
write('Ban muon cap nhat o don vi (?):');
read(tentep);
assign(f,tentep);
reset(f);
clrscr;
gotoxy(25,2);write('Chuong trinh cap nhat');
gotoxy(25,2);
BEGIN
write('Cap nhat can bo co ma so?(0:Dung lai):');
readln(n);
while (n>=1) and (n<=Filesize(f)) do
BEGIN
seek(f,n-1);
read(f,nguoi);
x:=12;
y:=10;
with nguoi do
BEGIN
gotoxy(x,y);
write('Mcb:',macb);gotoxy(x,y+1);
```

```
  write('Ten:',ten);clreol;gotoxy(x,y+2);
  write('HsL:',hsl);clreol;gotoxy(x,y+3);
  write('PhT:',pht);clreol;
  gotoxy(x+39,y+1);
  readln(st);
  if st<>' ' then
  BEGIN
  ten:=st;
  gotoxy(x,y+1);
  write('ten:',ten);clreol;
  END;
  gotoxy(x+15,y+2);
  readln(st);
  if st<> ' ' then
  BEGIN
  Val(st,i,error);
  if error=0 then hsl:=i;
  END;
  gotoxy(x,y+2);
  write('Hsl:',hsl);clreol;
  clreol;
  gotoxy(x+18,y+3);
  readln(st);
  if st<> ' ' then
  BEGIN
  Val(st,R,error);
  if error=0 then pht:=R;
  END;
  gotoxy(x,y+3);
  write('PhT:',pht);clreol;
  END;
  if nguoi.ten <> ' ' then
```

```
BEGIN
seek(f,n-1);
write(f,nguoi);
END;
gotoxy(25,2);
write('Cap nhat nguoi co so thu tu?(0=dung lai) ');
readln(n);
END;
END;
close(f);
END;
{***********}
procedure infile;
Var
f:file of canbo;
nguoi:canbo;
k:char;
i:integer;
tong2,tong,luong,dl:longint;
tentep:string[14];
BEGIN
clrscr;gotoxy(5,10);
write('In don vi(?):');
readln(tentep);
textbackground(13);
assign(f,tentep);
reset(f);
clrscr;
gotoxy(9,3);
textcolor(YELLOW);
write('**KET QUA IN BANG LUONG CUA DON VI:',TENTEP,'**');
gotoxy(4,4);    TEXTCOLOR(11);
```

```
write('-----------------------------------');
gotoxy(4,5);
write('|Ma cb|  Ho & Ten |He so| Luong  |Phai tru|  Duoc linh|');
gotoxy(4,6);
write('-----------------------------------');
gotoxy(4,7);
        tong2:=0;
        tong:=0;
        for i:=1 to filesize(f) do
        BEGIN
        seek(f,i-1);
        read(f,nguoi);
        With nguoi do
        BEGIN
        luong:=144000*hsl;
        dl:=luong-pht;
        gotoxy(4,i+6);
writeln('|',macb:5,'|',ten:20,'|',hsl:7,'|',luong:12,'|',pht:10,'|',dl:12,'|');
        END;
          tong:=tong+nguoi.luong;
            tong2:=tong2+nguoi.dl;
              END;
              gotoxy(4,7+filesize(f));
      write('-----------------------------------');
      gotoxy(4,8+filesize(f));
textcolor(red);highvideo;
write('*Tongcong:      ',tong,'   ',tong2);
readln;
END;
{**********************************}
procedure TongHop;
var
```

```
tdv: array[1..20] of string;
l:array[1..20] of longint;
pt:array[1..20] of longint;
dlinh:array[1..20] of longint;
n,x,y,h,t,i: integer;
Tk,tendv:string;
f:file of canbo;
nguoi: canbo;
begin
clrscr;
x:=2;
y:=2;
gotoxy(x,y); write('Ten khoa can bao cao:'); readln(tk);
gotoxy(x,y+1); write('So don vi trong khoa:'); readln(n);
h:=1;
while h<=n do
BEGIN
    gotoxy(x+16, y+3);
    write('      ');
    gotoxy(x,y+3);
    textcolor(h);
    Write('Nhap ten don vi:');
    readln(tendv);
    tdv[h]:= tendv;
    assign(f,tendv);
    reset(f);
    l[h]:=0; pt[h]:=0;  dlinh[h]:=0;
    for t:=1 to filesize(f) do
    BEGIN
    seek(f,t-1);
    read(f,nguoi);
    with nguoi do
```

```
        begin
     luong:=hsl*144000;
     dl:=luong - pht;
        end;
     l[h]:=l[h]+nguoi.luong;
     pt[h]:=pt[h]+nguoi.pht;
     dlinh[h]:=dlinh[h] + nguoi.dl;
     END;
       h:=h+1;
  END;
BEGIN
    clrscr;
    textcolor(yellow);
    gotoxy(x+12,y);
    write('****TONG HOP BANG LUONG CUA KHOA:',TK,'****');
    gotoxy(x+3,y+1);
    write('_____');
    gotoxy(x+3,y+2);
    write('|MaSo|Ten dv|Tong tien luong|Tong phai tru  |Tong duoc linh|');
    gotoxy(x+3,y+3);
     write('_____');
     for i:=1 to n do
     BEGIN
     gotoxy(x+3,y+3+i);
     write('|',i:4,'|',tdv[i]:8,'|',pt[i]:19,'|',dlinh[i]:18,'|');
     END;
     gotoxy(x+3,y+4+i);
     write('_____');
     readln;
  END;
     END;
     {**************************************}
```

```
BEGIN
    textbackground(13);
    REPEAT;
    clrscr;
    textcolor(11);
    gotoxy(15,7);
    writeln('2: SUA  SO LIEU');
    gotoxy(15,9);
    writeln('3: IN BANG LUONG TUNG DON VI');
    gotoxy(15,10);
    writeln('4:IN BANG LUONG CUA KHOA');
    gotoxy(13,15);
    write('can than(!) khi An so<1>');
    gotoxy(13,16);
    write('Vi file du lieu cung ten dang ton tai tren dia cua ban se mat');
    gotoxy(13,16);
    write(13,17);
    write('An<k> de thoat ve DOS');
    i:=readkey;
    if i='2' then cap_nhat;
    if i='1' then tao_file;
    if i='3' then infile;
    IF i='4' then tonghop;
    until i='k';
    END.
```

II. CHƯƠNG TRÌNH SƠ ĐỒ MẠNG - SỬ DỤNG CẤU TRÚC ARRAY

Bộ chương trình SĐM.01 và SĐM.03 được viết trên ngôn ngữ FORTRANIV năm 1980 - 81, chạy tính trên máy tính Minsk - 32. Đã chạy tính thử nghiệm đề tài tốt nghiệp của sinh viên khoa Cầu Đường khóa 23 (thiết kế Cầu Niệm, Hải Phòng) với kết quả tốt. Để phục vụ việc lập kế hoạch và quản lý thi công xây dựng có hiệu quả hơn, năm 1987 chúng tôi đã cài đặt lại các chương trình trên trong chế độ hội

thoại người máy bằng ngôn ngữ FORTRAN - 77 trên máy vi tính. Với những chương trình này chúng ta có thể kịp thời xây dựng kế hoạch ngay sau khi có hồ sơ thiết kế. Đến năm 1992, các thuật toán trên đã được viết chạy trên ngôn ngữ FOXPRO trong cho hệ thống chương trình "Trợ giúp lập kế hoạch sản xuất kinh doanh xây dựng", sẽ được giới thiệu dưới đây.

```
* CHUONG TRINH TO CHUC THI CONG CONG TRINH THEO SO DO MANG*
*= = = = = = = = = = ==========================*
set date french
set talk off
close data
clea
voothicong = 1
do while voothicong = 1
    anhthicong = 8
    * set color to / +b, /+b, +g
    clear
    set color  to +w* / +b, +r/ +b+, g+
    @ 1, 30 say " LUA CHON :"
    set color to +w / +b, +r / +b+, g+
    @ 3, 26 say " CONSTRUCTION EXECUTION "
    set color to +r /+g, +W/ bg+, g+
    @ 5,6 to 7, 30
    @ 6, 6 PROMPT "  QUAN  LY FIIES              "
    @ 5, 38 to 7, 65
    @ 6, 38 prompt " NHAP SO LIEU SO DO MANG   "
    @ 8, 6 to 10, 30
    *   set color to +g / +b, +rb / bg+, g+
    @ 9,6 prompt "  TINH THAM SO THOI GIAN "
    *  set color to +gr / +b, +W / +R
    @ 8, 38 to 10, 65
    * set color to +g / +b, +rb / bg+, g+
```

```
@ 9,38 prompt "  TINH VAT TU                    "
* set color to +gr /+b, +w / +r
@  11,6 to 13,30
* set color to +g / +b, +rb / bg+, g+
@  12,6 prompt  " IN DUONG GANG  "
*  set color to +rg / +b, +w / +r
@ 11,38 to 13,65
*  set color to +g / +b, +rb / bg+,g+
@12,38 prompt  "  CHINH LY SO DO MANG   "
*   set color to +gr / +b, +w / +r
@  14,6 to 16,30
*  set color to  r+ / b, rb+ / bg+
@  15,6 prompt  "  IN BANG BIEU  "
*  set color to +br / +b, +w /+r
@14,38 to 16,65
*  set color to w+ / b, w+ / r, g
@  15,38 prompt  " VE MENU CHINH   "
set color to +gr / +b, +w / +r
@ 18,26 SAY  "  MENU TO CHUC THI CONG  "
menu to anhthicong
@ 18,19 say "  = = = = = = = = = = = = = = = = = = = =  "
set color to +w* / b,w+ / r,g
@  20,20 say " DANG LAM VIEC "
@ 21,1 say "
"
@ 22,1 say "
"
@ 23,1 say "
"
@ 24,1 say "
"
do case
        case anhthicong = 1
```

```
            do qlf
            * = = = == = = = = = = = = = = = = = = = = = = = = = = = = *
case anhthicong = 2
            close data
            et talk off
             vaosl = 'Y'
            do while upper (vaosl) = 'Y'
                        set color  to  +r / +g, / +g,+b
                        set date british
                        clear
                        set color to +r / +g, / +g,+b
                        anthicong = 2
                        set color  to +W / +b, +r / +b,+b
                  @  2,24 say  "CHUONG TRINH NHAP SO LIEU  "
                  @  3,24 say  "  BANG THAM SO DO MANG   "
                  @  7,15 prompt  " 1. Vao cac tham so thoi gian  "
                        @  9,15 prompt   "  Ket thuc    "
                        menu to anthicong
                        do case
                                    case anthicong = 1
                                    tc0 = 1
                                    thtc0 = '                '
                                    clear
@  6,10 say  " DANG VAO CAC THAM SO THOI GIAN  "
                        @  10,10 say  " Hay chon ten file du lieu  "
                        @  11,30 to 17,45 double
                        @  12,36 prompt  "mtcv"
                        @  14,36 prompt  "mtcv1"
                        @  16,36 prompt  "mtcv2"
                        menu to tc0
                        select 1
                        if tc0 = 1
```

```
                use mtcv
        else
                if tc0 = 2
                        use mtcv1
                else
                        if tc0 = 3
                                use mtcv2
                        endif
                endif
                                endif
        copy stru to tctg0
        select 2
        use tctg0
        vthicong = 'Y'
        do while upper(vthicong ) = 'Y'
                appe blan
                iMV = 0
                jMV = 0
                tijMV = 0
                tcv = '
                clea
                set color to +w /+r,+r/+b,+b
@ 1,4 say "CHU Y : dang vao cac tham so thoi gian"
                set color to +w/+b, +w/+r, +b
                @ 6,15 to 16,65
@ 8,17 say  " Dinh dau cong viec : " get iMV
@ 10,17 say " Dinh cuoi cong viec : " get jMV
@ 12,17 say " Ten cong viec " get tcv
@ 14,17 say " Thoi gian hoan thanh CV " get tijMV
                set color to +w/+b, r+/+b,+b
                READ
repl ski with iMV, skj with jMV, tij with tijMV, tcv with tc
```

```
                    skip
            acce  " Co vao tiep khong ? (Y/N)  " to vthicong
                    enddo && vthicong
* ------------------------------------------------------------------
                case anthicong = 2
                    exit
                endcase && theo anthicong
                clea
            acce  " Co vao so lieu tiep ( Y / N )  " to vaosl
            enddo  &&  vaosl
* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = *
            case anhthicong = 3
            set color to +w*/b,w+/r,g
            @ 20,20 say "  DANG LAM VIEC    "
    @ 21,1 say "                                                    "
    @ 22,1 say " Tinh cac tham so SO DO MANG                        "
    @ 23,1 say "                                                                "
    @ 24,1 say "                                                                "
            public MaxT, MinT,k
            set talk off
            set safe off
            set data brit
            close data
            sele 1
            use mtcv
            * zap
            *appe from SLBD
            go top
            scan for ski = 1
                repl Tkslj with 0 , Thsij  with Tij
            endscan
            go top
```

```
k = 0
do while  .not.eof ( )
        if (Tksij = 0) .and. (Thsij = 0 )
                store SKi  to k
                bght = recno()
                copy to cvtg1 for skj = k
                sele 2
                use cvtg1
                calculate Max(Thsij ) to maxT
                zap
                close data
                delete file cvtg1 . dbf
                sele 1
                use mtcv
                go bght
                repl Tksij with maxT , Thsij with Tksij + Tij
        endif
        if .not. eof()
                skip
        endif
enddo
close data
* Het qua tring nguoc, sang qua trinh thuan
sele 1
use mtcv
calcul Max(Thsij)  to Tgang
calcul Max(skj)  to slsk
repl all Thmij with Tgang, Tkmij with Thmij - Tij for skj = slsk
go bottom
do while .not. bof()
    if (Tkmij = 0 ) .and. ( Thmij = 0)
store SKj to k
```

```
                bght = recno()
                copy to CVTG2 for ski = k
                sele 2
                use cvtg2
                calculate min(Tkmij ) to minT
                close data
                delete file cvtg2.dbf
                sele 1
                use mtcv
                go bght
                repl Thmij with minT , Tkmij with Thmij - Tij
            endif
        if .not. bof()
                skip - 1
        endif
        enddo
        sele 1
        @ 20,10 say " An phim Ctrl - End de ra khoi Browse "
        brow
        close data
* = = == = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = *
    case anhthicong = 4
        DO TinhVT && Tinh vat tu
* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = *
    case anhthicong = 5
        close data
        use mtcv
        copy to critical for dttd = 0 .and. dttp = 0
        clea
        set color to +w/b, w+/r, g
        irg = 'Y'
        @ 21,10 say " Co in ra giay khong ?(Y/N) " get irg
```

```
            read
            If upper(irg)='Y'
                  clea
                  @ 15,15 say "   Hay chuan bi may in , giay ! "
                  set prin on
                  use critical
? chr(27) + "W1" + " CAC CONG VIEC GANG " + chr(27) + "W0"
                  list to print
                  set print off
                  use
                  else
                  use critical
    @ 23,15 say  " DANG HIEN CAC CONG VIEC GANG  "
                  disp all tcv
            endif
            use
                  case anhthicong = 6
            DO ChlySDM  &&  Chinh ly so do mang
* = = = = = = = = = = = = = ======= = = = = = = = = = = = = = = *
                  case anhthicong = 7
                        DO IN  &&  in bang bieu
* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = *
                  case anhthicong = 8
                        clea
                        @  5,62 prompt  " VE MENU CHINH "
            endcase  &&  anh thi cong
            clea
            input  " Co lam tiep khong ( 1/ 0 ) " to v00thicong
      enddo  &&  v00thicong
close data
return
```

III. CHƯƠNG TRÌNH QUY HOẠCH ĐỘNG - SỬ DỤNG CẤU TRÚC BẢN GHI (RECORD) VÀ FILE, MẢNG (ARRAY).

```
Program baitap_lon;
    uses   crt,dos;
    var
      ch,ch1: char;
      chon, chon1: integer;
      n,i: integer;
      e:array[0..10] of real;
      Etd: array[0..10,0..30] of real;
      hmax, hmin:array[0..10] of real;
      h:array[0..30,0..30] of real;
      g:array[1..10] of real;
      hc:array[0..10] of real;
      Eyc: real;
      D: real;
      truoc: array[1..10,0..30] of integer;
      truoch: array[1..10,0..30] of real;
      f: file of real;
      chiphi: array[0..10,0..30] of real;
Procedure nhap;
var i: integer;
begin
   write('Vao so lop cua mat duong'); readln(n);
   write('Vao cuong do nen dat e[0]'); readln(e[0]);
   write('vao cuong do yeu cau Eyc'); readln(Eyc);
   for i:=1 to n do
     begin
     write('Vao cuong do cua lop thu',i,' e[',i,']='); readln(e[i]);
     end;
  for i:=1 to n do
     begin
```

```
      write('Vao do day cuc dai thu',i,' '); readln(hmax[i]);
      write('Vao do day cuc tieu thu',i,' '); readln(hmin[i]);
      end;
   for i:=1 to n do
      begin
      write(' Vao gia thanh thu',i,' '); readln(g[i]);
      end;
      write(' vao ban kinh cua mat ep cung'); readln(D);
      end;
Procedure  catdulieu;
var  ch: char;
   i:integer;
   name: string;
begin
   write('Ban co muon cat du lieu khong(C/K)'); readln(ch);
   if upcase(ch)='c' then
   begin
      write('Ten file cat du lieu');
      readln(name);
      assign(f,name);
      rewrite(f);
      write(f,Eyc);
      write(f,e[0]);
      for i:=1 to n do write(f,e[i]);
      for i:=1 to n do write(f,hmax[i]);
      for i:=1 to n do write(f,hmin[i]);
      for i:=1 to n do write(f,g[i]);
      write(f,D);
    end;
end;
(* procedure max_min;
var i: integer;
```

```
    m:real;
  begin
     Etdmin[0]:=e[0];
     Etdmax[0]:=e[0];
     for i:=1 to n do
     begin
  i:=exp((e[i]/Etdmin[i-1])*2.5;
  min[i]:=Etdmin[i-1]/(1-(2/pi)*(1-1/exp(3.5*ln(m)))*arctan(m*hmin[i]/d);
  i:=exp((e[i]/Etdmax[i-1])*2.5);
  dmax[i]:=Etdmax[i-1]/(1-(2/pi)*(1-1/(exp(3.5*ln(m))))*arctan(m*hmax[i]/d);
     end;
  end;    *)
  procedure laydulieu;
  var     f:file of real;
          name:string;
  begin
     write('Vao ten file lay du lieu'); readln(name);
     assign(f,name);
     reset(f);
     n:=filesize(f);
     n:=trunc((n-3)/4);
     read(f,e[0]);
     read(f,Eyc);
     for i:=1 to n do read(f,e[i]);
     for i:=1 to n do read(f,hmax[i]);
     for i:=1 to n do read(f, hmin[i]);
     for i:=1 to n do read(f,g[i]);
     read(f,D);
end;
Procedure tinhtoan;
var lamda:real;
   i,j,k: integer;
```

```
  chi:array[0..30,0..30] of real;
  m:array[0..10,0..30] of real;
  min: real;
  Binh,Cuc,Ha,Hang,Nguyet:real;
begin
   lamda:=(Eyc-e[0])/30;
   for i:=1 to n-1 do
      for j:=0 to 30 do
         Etd[i,j]:=e[0]+j*lamda;
   Etd[0,0]:=e[0];
   Etd[n,30]:=Eyc;
   for j:=0 to 30 do
  begin
   m[1,0]:=exp((1/2.5)*ln(e[1]/e[0]));
   h[0,j]:=(D/m[1,0]*sin((1-e[0]/Etd[1,j])/((2/pi)*(1-1/exp(3.5*ln(m[1,0])))))/
   cos((1-e[0]/Etd[i,j])/((2/pi)*(1-1/(exp(3.5*ln(m[1,0])))));
   if (h[0,j]<=hmax[1]) and (h[0,j]>=hmin[1]) then chiphi[1,j]:=g[1]*h[0,j]
  else chiphi[1,j]:=900000000.0;
  end;
     for j:=0 to n do truoc[1,j]:=0;
     for j:=0 to n do truoch[1,j]:=h[0,j];
     for i:=2 to n-1 do
       begin
        for j:=0 to 30 do
           for k:=0 to 30 do
            begin
               m[i,j]:=exp(1/1.25*ln(e[i]/Etd[i-1,j]));
               Binh:= exp(3.5*ln(m[i,j]));
               Cuc:=(2/pi)*(1-1/Binh);
               Ha:= 1-Etd[i-1,j]/Etd[i,k] ;
               h[j,k]:=(D/m[i,j])*sin(Ha/Cuc)/cos(Ha/Cuc);
           if (h[j,k]>=hmin[i]) and (h[j,k]<=hmax[i]) then chi[j,k]:=g[i]*h[j,k]
```

```
                else
                chi[j,k]:=900000000;
            end;
           for k:=0 to 30 do
            begin
             min:=1000000000;
             for j:=0 to 30 do
      if (chiphi[i-1,j]+ chiphi[j,k]) < min then min:=chiphi[i-1,j] + chi[j,k] ;
            chiphi[i,k]:=min;
            for j:=0 to 30 do
            if chiphi[i,k]=chiphi[i-1,j] + chi[j,k] then
             begin
               truoc[i,k]:=j;
               truoch[i,k]:=h[j,k];
             end;
          end;
       end;
  for j:=0 to 30 do
   begin
       m[n,j]:= exp (1/1.25* ln(e[n]/Etd [n-1,j]));
       Binh:= exp(3.5*ln(m[n,j]));
       Cuc:=(2/pi)*(1-1/Binh);
       Ha:= 1-Etd[n-1,j]/Etd[j,30] ;
       h[j,30]:=(D/m[n,j])*Sin(Ha/Cuc)/Cos(Ha/Cuc);
       if (h[j,30]>=hmin[n])and(h[j,30]<=hmax[n]) then chi[j,n]:=g[n]*h[j,n]
         else chi[j,n]:=900000000;
   end;
   min:=1000000000;
   for j:=0 to 30 do
   if chiphi[n-1,j]+chi[j,n]<min then min:= chiphi[n-1,j]+chi[j,n];
   chiphi[n,30]:=min;
  for j:=0 to n do
```

```
  if chiphi[n,30] = chiphi[n-1,j]+chi[j,30] then
      begin
      truoc[n,30]:=j;
      truoch[n,30]:=h[j,30];
      end;
      writeln('chi phi nho nhat la ',chiphi[n,30]);
      j:=truoc[n,30];hc[n]:=truoch[n,30];
      for i:=n-1 downto 1 do
      begin
      hc[i]:=truoc[i,j];
      j:=truoc[i,j];
end;
end;
          {CHUONG TRINH CHINH}
BEGIN
   repeat
   clrscr;
   textbackground(white);
   clrscr;
   textcolor(yellow);
   window(3,2,78,24);
   textbackground(blue);
   clrscr;
   chon:=1;
   ch:= #80;
   window(25,11,55,15);
   writeln('Menu chon lua');
   writeln('1.nhap du lieu');
   writeln('2.Tinh toan');
   writeln('3.Thoat khoi chuong trinh');
   repeat
      if (chon=1) and (ch=#80) then
```

```
    begin
     window(24,14,55,14);
     textbackground(blue);
     write('3. thoat khoi chuong trinh');
     window(24,12,55,12);
     textbackground(red);
     write('1. Nhap du lieu');
    end;
  if (chon=1)and (ch=#72) then
   begin
    window(24,13,55,13);
    textbackground(blue);
    write('2. tinh toan');
    window(24,12,55,12);
    textbackground(red);
    write('1. Nhap du lieu');
   end;
 if (chon=2) and (ch=#80) then
  begin
   window(24,12,55,12);
   textbackground(blue);
   write('1. Nhap du lieu');
   window(24,13,55,13);
   textbackground(red);
   write('2. Tinh toan');
  end;
 if (chon=2) and (ch=#72) then
  begin
   window(24,14,55,14);
   textbackground(blue);
   write('3. thoat khoi chuong trinh');
   window(24,13,55,13);
```

```
      textbackground(red);
      write('2. Tinh toan');
     end;
   if (chon=3) and (ch=#80) then
    begin
     window(24,13,55,13);
     textbackground(blue);
     write('2.Tinh toan');
     window(24,14,55,14);
     textbackground(red);
     write('3. Thoat khoi chuong trinh');
    end;
   if (chon=3) and (ch=#72) then
    begin
     window(24,12,55,12);
     textbackground(blue);
     clrscr;
     write('1. Nhap du lieu');
     window(24,14,55,14);
     textbackground(red);
     write('3. Thoat khoi chuong trinh');
    end;
   ch:=readkey;
   if ch=#80 then
    if chon<3 then chon:=chon+1 else chon:=1;
   if ch=#72 then
    if chon>1 then chon:=chon-1 else chon:=3;
until ch = #13;
if chon=1 then
  begin
  window(3,2,78,24);
  textbackground(blue);
```

```
    clrscr;
    nhap;
    catdulieu;
    end;
  if  chon=2 then
    begin
    window(3,2,78,24);
    textbackground(blue);
    clrscr;
    window(25,12,55,12);
    write(' Chon cach vao du lieu');
    chon1:=1;
    window(25,14,55,14);
    textbackground(blue);
    clrscr;
    write('2. Vao du lieu tu file');
    window(25,13,55,13);
    textbackground(red);
    write('1. Vao du lieu tu ban phim');
    repeat
        ch:=readkey;
        if (ch=#80) or (ch=#72) then
          if chon1=1 then chon1:=2 else chon1:=1;
        if chon1=1 then
          begin
          window(24,14,55,14);
          textbackground(blue);
          clrscr;
          write('2. Vao du lieu tu file');
          window(24,13,55,13);
          textbackground(red);
          write('1. Vao du lieu tu ban phim');
```

```
            end
          else
            begin
            window(25,13,55,13);
            textbackground(blue);
            clrscr;
            write('1. Vao du lieu tu ban phim');
            window(24,14,55,14);
            textbackground(red);
            write('3. Vao du lieu tu file');
            end;
 until  ch=#13;
end;
   if chon=1 then
     begin
        window(3,2,78,24);
        textbackground(blue);
        clrscr;
        nhap;
     end;
   if chon1=2 then
      begin
        window(3,2,78,24);
        textbackground(blue);
        clrscr;
        laydulieu;
      end;
      tinhtoan;
      window(3,2,78,24);
      textbackground(blue);
      clrscr;
      write('Be day cua cac lop mat duong');
```

```
        for i:=1 to n do
          write('h[',i,']=',hc[i]:4:2);
          writeln;
          writeln('chi phi nho nhat la',chiphi[n,30]);
      readln;
  until  chon=3;
 END.
```

IV. CHƯƠNG TRÌNH QUẢN LÝ THI TRẮC NGHIỆM - SỬ DỤNG CẤU TRÚC BẢN GHI (RECORD) VÀ FILE, MẢNG (ARRAY)., CON TRỎ (POINTER)

```
Program Trac_nghiem_pascal;
uses Crt,graph;
type
   q=^nut;
  string100=string[250];
  string30=string[250];
  nut=record
   cauhoi:string100;
   da:array[1..4] of string30;
   tl:1..4;
   ds:boolean;
   lk:q;
   end;
  Var
  a:nut;
  f:text;
  i:integer;
  d1,c1,d2,c2:q;
  a1,b1,a2,b2:integer;
  b:char;
  {------------------------}
```

```
procedure gwrite(var x,y:integer;gtext:string);
{viet dong text ra man hinh do hoa tai toa do x,y}
begin
 outtextxy(x,y,gtext);
 x:=x+textwidth(gtext);
End;
{--------------------}
procedure gwriteln(var x,y:integer;gtext:string);
{nhu gwrite nhung xuong dong }
begin
  outtextxy(x,y,gtext);
  y:=y+textheight('H')+10;
  x:=110;
End;
{------------------------------}
procedure greadln(t:byte;var x,y:integer;var gtext:string);
  {doc vao dong text va xuong dong}
  const enter=#13;xoa=#8;
  var ch,ch1: char;
      i:integer;
  begin
   gtext:=' ';
   while (ch<>enter) and (length(gtext) < t) do
   begin
   ch:=readkey;
    if (ch<>enter)and(ch<>xoa)then
     begin
      ch1:=ch;
      setcolor(5);
      gwrite(x,y,ch);
      gtext:=gtext+ch;
     end
```

```
    else if (ch=xoa) and (length(gtext)>0) then
    begin
     i:=length(gtext)-1;
     x:=x-textwidth(ch);
     setcolor(lightgreen);
     gwrite(x,y,gtext[i+1]);{xoa}
     x:=x-textwidth(ch);
     gtext:=copy(gtext,1,i);
    end;
   end;
   if(length(gtext))=t then readln;
   setcolor(11);
   x:=110;y:=y+textheight('H')+20;
End;
{-------------------------------------}
procedure remove(d,c:q);
Var p:q;
begin
  while d<>nil do
  begin
   p:=d;
   d:=d^.lk;
   dispose(p);
  end;
end;
{-------------------------------}
procedure initq(var d1,c1,d2,c2:q);
begin
 d1:=nil;
 c1:=nil;
 d2:=nil;
 c2:=nil;
```

```
end;
{---------------------------------}
procedure MhDohoa;
var Gd,Gm:integer;
   path:string[50];
begin
   Gd:=detect;
  { path:='c:\tp\bgi';}
   path:='d:\tp(full)\bgi';
   initgraph(gd,gm,path);
   while graphresult<>0 do
  begin
   write('Loi do hoa, hay go duong dan toi*.BGI, hoac ENTER de thoat');
   readln(path);
   if (path= ' ') then halt(1);
    Gd:=Detect;
   initgraph(gd,gm,path);
  end;
End;
{---------------------------------}
procedure Cua_so(c1,h1,c2,h2,v,m_t,m_d,m_n:integer);
Var
  i:integer;
Begin
 for i:=1 to v do
 begin
  Setcolor(m_t);
  line(c1+i,h1+i,c2-i,h1+i);{Tren}
  Line(c1+i,h1+i,c1+i,h2-i);{Trai}
  Setcolor(m_d);
  line(c1+i,h2-i,c2-i,h2-i);{duoi}
  Line(c2-i,h2-i,c2-i,h1+i);{phai}
```

```
 SetFillStyle(1,m_n); Bar(c1+v,h1+v,c2-v,h2-v);
 end;
 End;
 {--------------------------------}
 procedure help(x1,y1,x2,y2:integer;muc:integer);
 var bitmap:pointer;
    size:word;
    ch:char;
 Begin
  size:=imagesize(x1,y1,x2,y2);
  getmem(bitmap,size);
  getimage(x1,y1,x2,y2,bitmap^);
 case muc of
  1:begin
  cua_so(x1,y1,x2,y2,2,white,white,blue);
  settextstyle(1,0,1);
  setcolor(11);
  outtextxy(x1+40,y1+5,'          Giup do!');
  Settextstyle(0,0,1);
  setcolor(lightgreen);
 outtextxy(x1+5,y1+45,' -Day la chuong trinh thi trac nghiem pascal!');
    setcolor(white);
 outtextxy(x1+5,y1+textheight('A')+60,' -De su dung hay dung phim');
 outtextxy(x1+5,y1+textheight('A')+75,' -De chon muc can su dung');
 outtextxy(x1+5,y1+textheight('A')+90,' -sau khi chon duoc muc ');
  outtextxy(x1+5,y1+textheight('A')+105,' - an nut enter de lam');
  outtextxy(x1+5,y1+textheight('A')+125,' - thoat ra khoi menu');
  outtextxy(x1+5,y1+textheight('A')+140,' -chon Exit or Esc de thoat');
    setcolor(lightred);
    outtextxy(x1+5,y1+textheight('A')+180,'  -Bam mot nut bat ky de tro lai
chuong trinh');
   end;
```

```
2:begin
  cua_so(x1,y1,x2,y2,2,white,white,1{blue});
  settextstyle(1,0,1);
  setcolor(11);
  outtextxy(x1+40,y1+5,'              Giup do!');
  Settextstyle(0,0,1);
  setcolor(lightgreen);
  outtextxy(x1+5,y1+45,' -Day la chuong trinh thi trac nghiem pascal!');
  setcolor(white);
  outtextxy(x1+5,y1+textheight('A')+60,'  -De su dung ban hay dung phim');
  outtextxy(x1+5,y1+textheight('A')+75,'  -De lua chon cau tr loi thich hop');
  outtextxy(x1+5,y1+textheight('A')+90,' -sau khi chon duoc cau tra loi thich');
  outtextxy(x1+5,y1+textheight('A')+105,' -hop ban hay bam enter');
  outtextxy(x1+5,y1+textheight('A')+120,' -hoac ban co the su dung cac phim');
  outtextxy(x1+5,y1+textheight('A')+135,' -(A),(B),(C),(D) de lua chon');
  setcolor(lightred);
  outtextxy(x1+5,y1+textheight('A')+155,' -An nut bat ky de tro ve
                                                    chuong  trinh');

  end;
3:begin
  cua_so(x1,y1,x2,y2,2,white,8,7);
  cua_so(x1+5,y1+5,x2-5,y2-5,2,1{blue,blue},1,red);
  setcolor(11{yellow});
  outtextxy(x1+10,y1+50,'Ban co muon thoat khong?(c/k):');
  end;
 end;
 b:=readkey;
 putimage(x1,y1,bitmap^,copyput);
 freemem(bitmap,size);
end;
```

```
{-----------------------------------------}
procedure nen_man_hinh;
begin
   setfillstyle(9,15);
   bar(0,0,getmaxx,getmaxy);
   cua_so(90,30,550,153,1,0,0,1);{bong den}
   cua_so(80,20,540,140,2,white,white,lightblue);
   settextstyle(1,0,1);
   for i:=1 to 3 do
    begin
     setcolor(black);
     outtextxy(130+i,40-i, 'THI TRAC NGHIEM HE DIEU HANH
                                                    DOS+PASCAL');
     outtextxy(120+i,80-i,'Bo mon Tin hoc Truong Dai Hoc Xay Dung');
    end;
     setcolor(14);
     outtextxy(130,40,'THI TRAC NGHIEM HE DIEU HANH
                                                    DOS+PASCAL');
     outtextxy(120,80,'Bo mon Tin hoc Truong Dai Hoc Xay Dung');
     settextstyle(0,0,1);
     setcolor(white);
     setfillstyle(solidfill,lightblue);
     bar(0,460,640,480);
     setcolor(11);
     outtextxy(40,470,'An F1 de Giup do');
     outtextxy(160,470,'An ESC de Thoat');
   end;
   {-----------------------------------------}
   procedure ve_menu(p:q; chon:integer);{hien cac dap an tra loi}
   Var
     i:integer;
     mang_mn:array[1..4] of string;{dung luu cac dong menu}
```

```
begin
 for i:=1 to 4 do mang_mn[i]:=p^.da[i];
   Settextstyle(0,0,1);
   for i:=1 to 4 do
    begin
       {neu chon =i thi dong i to mau khac}
      if i= chon then
       cua_so(20,190+i*30,605,215+i*30,1,lightblue,lightblue,green)
        else
       cua_so(20,190+i*30,605,215+i*30,1,lightblue,lightblue,lightblue);
       {Viet dong chu vao hinh chu nhat}
       setcolor(white);
       outtextxy(20,200+i*30,mang_mn[i]);
    end;
end;
{-----------------------------------}
procedure Chon_Doc(var ch:integer;p:q;var done:boolean);{chon cau
tra loi}
Var
  Sott: integer;
  key: char;
  begin
  Settextstyle(0,0,1);
  cua_so(20,70,620,100,1,lightblue,lightblue,lightblue);
 setcolor(11);
 outtextxy(20,80,p^.cauhoi);
 sott:=1;
 ve_menu(p,Sott);
 while not done do
    begin
      key:=readkey; {dieu khien phim chon menu}
      case upcase(key) of
```

```
            #13:
             case Sott of
                1: begin
                     ch:=1;
                     exit;
                   end;
                2: begin
                     ch:=2;
                     exit;
                   end;
                3: begin
                     ch:=3;
                     exit;
                   end;
                4: begin
                     ch:=4;
                     exit;
                   end;
           end;
          #72:
            Begin
             if Sott= 1 then Sott:=4
               else
                 Sott := Sott-1;
            end;
#80:
  begin
     Sott:=Sott+1;
     if Sott>4 then Sott:=1;
  end; {go so cung duoc !}
#65: begin
     ch:=1;
```

```
        {#65: ma ASCII cua chu A}
        exit;
      end;
#66: begin
        ch:=2;
        exit;
     end;
#67: begin
        ch:=3;
        exit;
      end;
#68: begin
        ch:=4;
         exit;
      end;
  #27: begin  {neu nhan esc thi quay tro ve menu chinh }
        help(200,100,500,170,3);
        b:=upcase(b);
        if b='C' then
          begin
             remove(d1,c1);
             remove(d2,c2);
             initq(d1,c1,d2,c2);
             close(f);
             nen_man_hinh;
             done:= true;
           end;
        end;
   #59: help(a1,b1,a2,b2,2);
  end;
   {of case key}
   {Tro ve Menu}
```

```
            if not done then ve_menu(p,sott);
          end; {of while}
        End;
            {--------------------------------}
      procedure addq(var d,c:q;x:nut); {themmot gia tri vao queue}
        var
          p:q;
          i: integer;
        begin
              new(p);
              p^.cauhoi:=x.cauhoi;
              for i:=1 to 4 do
              p^.da:=x.da;
              p^.tl:=x.tl;
              p^.ds:= false;
              p^.lk:= nil;
              if d= nil then
               begin
                 d:=p;
                 c:=p;
               end
              else
               begin
                 c^.lk:=p;
                 c:=p;
               end;
        end;
              {--------------------------}
      procedure duyetq(d,c:q;var done: boolean; ten:string; maso:string);
{duyet queue}
         var p:q;
            ch,diem, count1, count2: integer;
```

```
            x:nut;
            tg: string;
            a: char;
            begin
                diem:=0;
                p:=d;
                count1:=0; count2:=0;
                while p<>nil do
                begin
                chon_Doc(ch,p,done);
                if done then exit;
                if ch=p^.tl then
                 begin
                    diem:= diem+1;
                    p^.ds:=true;
                    count1:= count1+1;
                 end
                 else
                   begin
                     x:= p^;
                     addq(d2,c2,x);
                   end;
                 p:=p^.lk;
              end;{of while}
             p:=d2;
            {In ra bang ket qua}
cleardevice;
Cua_so(30,100,600,400,3,white,8,7);
Cua_so(40,110,590,390,2,white,8,7);
cua_so(45,115,585,165,2,8,white,1{blue});
settextstyle(1,horizdir,5);
for i:=1 to 3 do
```

```
    begin
      setcolor(black);
      outtextxy(80+i,115-i,'Ket Qua Kiem Tra');
    end;
  setcolor(11);
  outtextxy(80,115,'Ket Qua Kiem Tra');
  cua_so(45,200,585,385,2,8,white,5);
  settextstyle(1,horizdir,1);
  setcolor(11);
  outtextxy(180,210,'Sinh vien  :');
  outtextxy(180+textwidth('sinh  vien  :'),210,ten);
  setcolor(11);
  outtextxy(180,230,'Ma so sinh vien :');
  outtextxy(180+textwidth('Ma so sinh vien  :'),230,maso);
  setcolor(11);
 outtextxy(60,250,'=====================================');
  setcolor(white);
  outtextxy(80,300,'So cau tra loi dung:');
  str(count1,tg);outtextxy(80+textwidth('So cau tra loi dung:'),300,tg);
  outtextxy(80,330,'Tong so diem  :');
  str(diem,tg);outtextxy(80+textwidth('tong so diem  :'),330,tg);
 remove(d1,c1);
 remove(d2,c2);
 close(f);end;
 {---------------------------------------}
 procedure doc_file(var a:nut);
   var
     i:integer;
   begin
      readln(f,a.tl,a.cauhoi);
      for i:=1 to 4 do readln(f,a.da[i]);
  end;
```

```
 {---------------------------------------}
procedure level(path:string;slch:integer;ten:string;maso:string);
  var
    done:boolean;
    i,j,m:integer;
  begin
      done:=false;
      cua_so(5,10,630,428,3,white,8,7);
      cua_so(13,20,622,50,2,8,8,lightblue);
      setfillstyle(solidfill,lightblue);
      bar(0,460,640,480);
      setcolor(11);
      outtextxy(40,465,'F1  Giup do');
      outtextxy(160,465,'ESC    Tro ve menu');
      Settextstyle(1,horizdir,1);
      for m:=1 to 3 do
         begin
         setcolor(red);
         outtextxy(160+m,22-m,'BAI THI TRAC NGHIEM HE DIEU HANH &
                                                          PASCAL');

         end;
       setcolor(14);
       outtextxy(160,22,'BAI THI TRAC NGHIEM HE DIEU HANH & PASCAL');
       cua_so(13,58,622,420,3,blue,blue,lightblue);
       Settextstyle(0,horizdir,1);
       initq(d1,c1,d2,c2);
       assign(f,path);
       reset(f);
       for j:=1 to 24 do
          begin
             doc_file(a);
             addq(d1,c1,a);
```

```
            end;
            duyetq(d1,c1,done,ten,maso);
          if done then exit;
          b:=readkey;
          nen_man_hinh;
       end;
{-----------------------------------------}
procedure mainmenu(chon:integer);
    Var
      i:integer;
      mang_mn:Array[1..2] of string;{dung luu cac dong menu}
    begin
     mang_mn[1]:='1. Phan bai thi';
     mang_mn[2]:='2 Thoat khoi chuong trinh';
     setcolor(white);
     Settextstyle(0,horizdir,1);
     for i:=1 to 2 do
       begin
       {Neu chon=i thi dong i to mau khac}
       if i=chon then
       cua_so(180,200+i*30,430,230+i*30,4,8,white,green)
        else
       cua_so(180,200+i*30,430,230+i*30,4,white,8,lightblue);
       {Viet dong thv vao hinh chu nhat}
       setcolor(14);
       outtextxy(190,210+i*30,mang_mn[i]);
     end;
  end;
{-----------------------------------------}
procedure dan_nhap(var ten,maso:string;var slch:integer);
 var
   x,y:integer;
```

```
  code:word;
begin
   cleardevice;
   cua_so(80,80,520,400,3,white,8,7);
   cua_so(95,95,505,385,2,white,8,7);
   cua_so(100,100,500,150,4,8,8,5);
   cua_so(100,180,500,380,4,8,8,5);
   Settextstyle(0,horizdir,1);
   setcolor(11);
   repeat
       x:=110; y:=230;
       gwrite(x,y,'Ho va ten  :');
       cua_so(x+10,y-6,x+200,y+10,1,1,1,lightgreen);
       x:=x+11;
       setcolor(1);
       greadln(25,x,y,ten);
    until length(ten)<25;
    x:=110; y:=230+textheight('H')+10;
    gwrite(x,y,'Ma so sinh vien :');
    cua_so(x+8,y-6,x+60,y+10,1,1,1,lightgreen);
    x:=x+11;
    setcolor(5);
    greadln(6,x,y,maso);
    val(maso,slch,code);
    cleardevice;
  end;
{-------------------------------------}
   Procedure chonmain;
     Var Sott:integer;
        key:char;
        slch:integer;
        maso,ten:string;
```

```
Begin
Sott:=1;
slch:=0;
mainmenu(Sott); { Ve hinh chu nhat to mau muc 1 }
While true do   {lap khong dieu kien }
  begin
    key:=readkey; {dieu khien phim chon menu}
    case key of
     #13:
       case Sott of
         1:begin
             cleardevice;
             dan_nhap(ten,maso,slch);
             level('cauhoi.txt',slch,ten,maso);
           end;
         2:exit;
       end;
    #27:exit;
    #72:
      Begin
       Sott:=Sott-1;
       if Sott<1 then Sott:=2;
      End;
    #80:
      Begin
        Sott:=Sott+1;
        if Sott>2 then Sott:=1;
      End;
   #59:help(a1,b1,a2,b2,1)
end; { of case key }
{Tro ve MENU}
mainmenu(Sott);
```

```
    end;{of While}
  end;
BEGIN {Chuong trinh chinh}
    mhdohoa;
    setbkcolor(5);
    a1:=100; b1:=50; a2:=500; b2:=300;
    setbkcolor(6);
    nen_man_hinh;
    chonmain;
    closegraph;
END.
```

TÀI LIỆU THAM KHẢO

[1] N.Wirth. *Cấu trúc dữ liệu + thuật giải = chương trình*

[2] Larry Nyhoff, Sanford Leestma. *Lập trình nâng cao bằng Pascal với các cấu trúc dữ liệu*. Bản dịch của PTS Lê Minh Trung.

[3] Nguyễn Mậu Bành. *Đề tài 28B-02-02 "Hoàn thiện công tác kế hoạch hoá xây dựng cơ bản ở Việt Nam"*. Hà Nội, 1991.

[4] Nguyễn Văn Chọn. *So sánh phương án và tính toán hiệu quả kinh tế trong cơ giới hoá xây dựng*. ĐHXD, Hà Nội, 1987.

[5] Bùi Công Cường. *Khoa học hệ thống và việc xây dựng các hệ trợ giúp soạn thảo ra quyết định*. Thông tin KHKT, tập Toán - cơ - khoa học tính toán. Viện KHVN, số 1/1990.

[6] Bùi Công Cường, Hoàng Nghĩa Tý. *Hệ trợ giúp quyết định kế hoạch cho xí nghiệp*. Báo cáo Hội nghị Toán học toàn quốc, Hà Nội, 9/1990.

[7] FOXPRO. *Tài liệu dịch từ bản gốc của Công ty Phần mềm FoxSoftware*. Hà Nội 1991.

[8] Hoàng Nghĩa Tý. *Thuật toán và chương trình sơ đồ mạng trong xây dựng*. Tuyển tập công trình khoa học, ĐHXD, số 1/1989.

[9] Hoàng Nghĩa Tý. *Thử nghiệm xây dựng hệ tin học trợ giúp quyết định cấp xí nghiệp*. Thông báo khoa học các trường đại học. 1991.

[10] Hoàng Nghĩa Tý. *Thử nghiệm xây dựng tin học trợ giúp quyết định kế hoạch*. Tuyển tập công trình khoa học của nghiên cứu sinh ĐHXD, tháng 11/1991.

[11] Hoàng Nghĩa Tý. *BASIC cho máy vi tính*. ĐHXD, 1991.

[12] Hoàng Nghĩa Tý. *Cơ sở tin học và ngôn ngữ lập trình Pascal*. Nxb KHKT, Hà Nội, 1991.

[13] Hoàng Nghĩa Tý, Phạm Thiếu Nga. *Giáo trình Tin học đại cương* Nxb KHKT, Hà Nội, 1995.

[14] ArisR. *Discrete Dynamic programming.* Blaisdell pud. Cor. 1964.

[15] Dal O.J., Dijkstra E.W., Hoare C.A.R. *Structured programming,* London, Academic press, 1972.

[16] Date C.j. *An introduction to Datebase Systems.*1975.

[17] Gries D. *The Science of Programming.* Springer - Verlag, 1981.

[18] Grogono P. *Programming in PASCAL.* Addison - Wesley P- Comp. 1980.

[19] Knuth D.E. *The Art of computer programming,* Volum I, II, III. Addison - Wesley, 1973.

[20] Teorey T.J., Fry J.P. *Design of Database structures.* Prentice-Hall, Inc. Engle Wood Cliffs, 1982.

[21] Tschritzis D.C.,Lochovsky F.H. *Data models.* Prentice- Hall, Inc. Engle Wood Ciffs, New Jersey 1982.

[22] With N. *Systematic Programming. An introduction.* Prentice - Hall Inc. EngleWood Ciffs, New jersey. 1973.

[23] Zadeh L.A. *The Concept of a linguistic variable and its applications to approximate reasoning.* Inf. Science, 1975-1976.

MỤC LỤC

CẤU TRÚC DỮ LIỆU VÀ THUẬT TOÁN

Chịu trách nhiệm xuất bản:
TRỊNH XUÂN SƠN

Biên tập:	ĐINH THỊ PHƯỢNG
Chế bản:	TRẦN KIM ANH
Sửa bản in:	ĐINH THỊ PHƯỢNG
Vẽ bìa:	NGUYỄN NGỌC DŨNG

In 300 cuốn khổ 17 × 24cm tại Xưởng in Nhà xuất bản Xây dựng. Giấy chấp nhận đăng ký kế hoạch xuất bản số 36-2013/CXB/140-158/XD ngày 5/01/2013. Quyết định xuất bản số 135-2013/QĐ-XBXD ngày 20/6/2013. In xong và nộp lưu chiểu tháng 8 năm 2013.