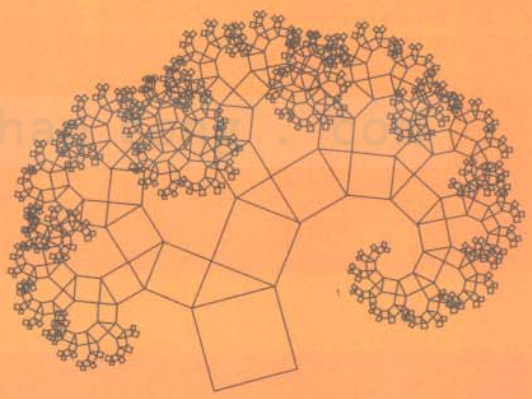
ĐỖ XUÂN LÔI

Cấu trúc dữ liệu và giải thuật



NHÀ XUẤT BẢN ĐẠI HỌC QUỐC GIA HÀ NỘI

ĐỖ XUÂN LÔI

Cấu trúc dữ liệu và giải thuật

(In lần thứ chín, có sửa chữa)

NHÀ XUẤT BẢN ĐẠI HỌC QUỐC GIA HÀ NỘI

NHÀ XUẤT BẢN ĐẠI HỌC QUỐC GIA HÀ NỘI

16 Hàng Chuối - Hai Bà Trưng - Hà Nội

Điện thoại: (04) 9718312; (04) 7547936. Fax: (04) 9714899

E-mail: nxb@ vnu.edu.vn

* * *

Chịu trách nhiệm xuất bản:

Giám đốc: PHÙNG QUỐC BÁO

Tổng biên tập: PHẠM THÀNH HƯNG

Biên tập: HỒ ĐỒNG

NGUYỄN TRỌNG HÀI

Trình bày bìa: SÁNG ĐỒNG

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Mã số: 1L-17 ĐH2006

In 1000 cuốn, khổ 16 x 24 cm tại Trung tâm In tranh Tuyên truyền cổ động - Mai Dịch, Cầu Giấy, Hà Nội

Số xuất bản: 85 - 2006/CXB/75 - 01/ĐHQGHN, ngày 24/01/2006.

Quyết định xuất bản số: 74 LK/XB

In xong và nộp lưu chiểu quý I năm 2006.

LỜI GIỚI THIỆU

(Cho lần xuất bản thứ bảy)

Kể từ năm 1993 đến nay, cuốn "Cấu trúc dữ liệu và giải thuật" của PGS. Đỗ Xuân Lôi đã được đông đảo bạn đọc đón nhận và hoan nghênh.

Cuốn sách này đã trở thành tài liệu học tập và tham khảo của sinh viên ngành Công nghệ Thông tin ở nhiều cơ sở đào tạo Cao đẳng, Đại học và sau Đại học.

Khác với 6 lần in trước, trong lần xuất bản này tác giả đã bổ sung và chỉnh lý lại nhiều phần. Tác giả cũng đã chú ý đúc rút kinh nghiệm qua nhiều năm giảng dạy để việc giới thiệu các nội dung kiến thức cũng như bài tập phù hợp hơn và dễ tiếp cận hơn đối với các đối tượng bạn đọc khác nhau.

Hy vọng rằng cuốn sách sẽ đáp ứng tốt hơn yêu cầu của bạn đọc trong việc nâng cao trình độ về công nghệ thông tin.

NHÀ XUẤT BẢN
ĐẠI HỌC QUỐC GIA HÀ NỘI

LỜI NÓI ĐẦU
(Cho lần xuất bản đầu tiên)

Cuốn sách này phản ánh nội dung của một môn học cơ sở trong chương trình đào tạo kỹ sư tin học. Ở đây sinh viên sẽ được làm quen với một số kiến thức cơ bản về cấu trúc dữ liệu và các giải thuật có liên quan, từ đó tạo điều kiện cho việc nâng cao thêm về kỹ thuật lập trình, về phương pháp giải các bài toán, giúp sinh viên có khả năng đi sâu thêm vào các môn học chuyên ngành như cơ sở dữ liệu, trí tuệ nhân tạo, hệ chuyên gia, ngôn ngữ hình thức, chương trình dịch v.v...

Nội dung cuốn sách được chia làm 3 phần

Phần I: Bổ sung thêm nhận thức về mối quan hệ giữa cấu trúc dữ liệu và giải thuật, về vấn đề thiết kế, phân tích giải thuật và về giải thuật đệ qui.

Phần II: Giới thiệu một số cấu trúc dữ liệu, giải thuật xử lý chúng và vài ứng dụng điển hình. Ở đây sinh viên sẽ tiếp cận với các cấu trúc như mảng, danh sách, cây, đồ thị và một vài cấu trúc phi tuyến khác. Sinh viên cũng có điều kiện để hiểu biết thêm về một số bài toán thuộc loại "phi số", cũng như thu lượm thêm kinh nghiệm về thiết kế, cài đặt và xử lý chúng.

Phần III: Tập trung vào "sắp xếp và tìm kiếm", một yêu cầu xử lý rất phổ biến trong các ứng dụng tin học. Có thể coi đây như một phần minh hoạ thêm cho việc ứng dụng các cấu trúc dữ liệu khác nhau trong cùng một loại bài toán.

Cuốn sách bao gồm 11 chương, chủ yếu giới thiệu các kiến thức cần thiết cho 90 tiết học, cả lý thuyết và bài tập (sau khi sinh viên đã học tin học đại cương). Tuy nhiên, với mục đích vừa làm tài liệu học tập, vừa làm tài liệu tham khảo, nên nội dung của nó có bao hàm thêm một số phần nâng cao.

Bài tập sau mỗi chương đã được chọn lọc ở mức trung bình, để sinh viên qua đó hiểu thêm bài giảng và thu hoạch thêm một số nội dung mới không được trực tiếp giới thiệu.

Cuốn sách có thể được dùng làm tài liệu học tập cho sinh viên hệ kỹ sư tin học, cử nhân tin học, cao đẳng tin học; làm tài liệu tham khảo cho sinh

viên cao học, nghiên cứu sinh, giảng viên tin học và các cán bộ tin học muốn nâng cao thêm trình độ.

Trong quá trình chuẩn bị, tác giả đã nhận được những ý kiến đóng góp về nội dung, cũng như các hoạt động hỗ trợ cho việc cuốn sách được sớm ra mắt bạn đọc. Tác giả xin chân thành cảm ơn GS. Nguyễn Đình Trí chủ nhiệm đề tài cấp nhà nước KC01-13 về Tin học - Điện tử - Viễn thông; PGS Nguyễn Xuân Huy, Viện tin học VN; PGS. Nguyễn Văn Ba, PTS Nguyễn Thanh Thủy và các đồng nghiệp trong Khoa Tin học trường ĐH Bách khoa HN.

Mặc dầu cuốn sách đã thể hiện được phần nào sự cân nhắc lựa chọn của tác giả trong việc kết hợp giữa yêu cầu khoa học với tính thực tiễn, tính sư phạm của các bài giảng, nhưng chắc chắn vẫn không tránh khỏi các thiếu sót. Tác giả mong muốn nhận được các ý kiến đóng góp thêm để có thể hoàn thiện hơn nữa nội dung cuốn sách, trong những lần tái bản sau.

Hà Nội ngày 15/8/1993

ĐỖ XUÂN LÔI

PHẦN I
GIẢI THUẬT

Chương 1

MỞ ĐẦU

1.1 Giải thuật và cấu trúc dữ liệu

Có thể, có lúc, khi nói tới việc giải quyết bài toán trên máy tính điện tử, người ta chỉ chú ý đến *giải thuật* (algorithms). Đó là một dãy các *câu lệnh* (statements) chặt chẽ và rõ ràng xác định một trình tự các thao tác trên một số đối tượng nào đó sao cho sau một số hữu hạn bước thực hiện ta đạt được kết quả mong muốn.

Nhưng, xét cho cùng, giải thuật chỉ phản ánh các phép xử lý, còn đối tượng để xử lý trên máy tính điện tử, chính là *dữ liệu* (data) chúng biểu diễn các thông tin cần thiết cho bài toán: các dữ kiện đưa vào, các kết quả trung gian... Không thể nói tới giải thuật mà không nghĩ tới: giải thuật đó được tác động trên dữ liệu nào, còn khi xét tới dữ liệu thì cũng phải hiểu: dữ liệu ấy cần được tác động giải thuật gì để đưa tới kết quả mong muốn.

Bản thân các phần tử của dữ liệu thường có mối quan hệ với nhau, ngoài ra nếu lại biết "tổ chức" theo các cấu trúc thích hợp thì việc thực hiện các phép xử lý trên các dữ liệu sẽ càng thuận lợi hơn, đạt hiệu quả cao hơn. Với một cấu trúc dữ liệu đã chọn ta sẽ có giải thuật xử lý tương ứng. Cấu trúc dữ liệu thay đổi, giải thuật cũng thay đổi theo. Ta sẽ thấy rõ điều đó qua ví dụ sau: Giả sử ta có một danh sách gồm những cặp "Tên đơn vị, số điện thoại": (a_1, b_1), (a_2, b_2), ..., (a_n, b_n).

Ta muốn viết một chương trình cho máy tính điện tử để khi cho biết "tên đơn vị" máy sẽ in ra cho ta: "số điện thoại". Đó là một loại bài toán mà phép xử lý cơ bản là "tìm kiếm".

- Một cách đơn giản là cứ điểm lần lượt các tên trong danh sách a_1, a_2, a_3 v.v... cho tới lúc tìm thấy tên đơn vị a_i nào đó, đã chỉ định, thì đối chiếu ra số điện thoại tương ứng b_i của nó. Nhưng việc đó chỉ làm được khi danh mục điện thoại ngắn, nghĩa là với n nhỏ, còn với n lớn thì rất mất thời gian.
- Nếu trước đó danh mục điện thoại đã được sắp xếp theo thứ tự tự điển

(dictionary order) đối với tên đơn vị, tất nhiên sẽ áp dụng một giải thuật tìm kiếm khác tốt hơn, như ta vẫn thường làm khi tra tự điển.

- Nếu tổ chức thêm một bảng mục lục chỉ dẫn theo chữ cái đầu tiên của "tên đơn vị", chắc rằng khi tìm số điện thoại của Đại học Bách khoa ta sẽ bỏ qua được các tên đơn vị mà chữ đầu không phải là chữ Đ.

Như vậy: giữa cấu trúc dữ liệu và giải thuật có mối quan hệ mật thiết. Có thể coi chúng như hình với bóng. Không thể nói tới cái này mà không nhắc tới cái kia.

Chính điều đó đã dẫn tới việc, cần nghiên cứu các *cấu trúc dữ liệu* (data structures) đi đôi với việc xác lập các giải thuật xử lý trên các cấu trúc ấy.

1.2 Cấu trúc dữ liệu và các vấn đề liên quan

1.2.1. Trong một bài toán, dữ liệu bao gồm một tập các phần tử cơ sở, mà ta gọi là *dữ liệu nguyên tử* (atoms). Nó có thể là một chữ số, một ký tự... nhưng cũng có thể là một con số, hay một từ..., điều đó tuỳ thuộc vào từng bài toán.

Trên cơ sở của các dữ liệu nguyên tử, các cung cách (manners) khả dĩ theo đó liên kết chúng lại với nhau, sẽ dẫn tới các cấu trúc dữ liệu khác nhau.

Lựa chọn một cấu trúc dữ liệu thích hợp để tổ chức dữ liệu vào và trên cơ sở đó xây dựng được giải thuật xử lý hữu hiệu đưa tới kết quả mong muốn cho bài toán, đó là một khâu rất quan trọng.

Cần chú ý rằng, trong những năm gần đây, lớp các khái niệm về cấu trúc dữ liệu đã tăng lên đáng kể. Thoạt đầu, khi ứng dụng của máy tính điện tử chỉ mới có trong phạm vi các bài toán khoa học kỹ thuật thì ta chỉ gặp các cấu trúc dữ liệu đơn giản như biến, vectơ, ma trận v.v... nhưng khi các ứng dụng đó đã mở rộng sang các lĩnh vực khác mà ta thường gọi là các *bài toán phi số* (non-numerical problems), với đặc điểm thể hiện ở chỗ: khối lượng dữ liệu lớn, đa dạng, biến động; phép xử lý thường không phải chỉ là các phép số học... thì các cấu trúc này không đủ đặc trưng cho các mối quan hệ mới của dữ liệu nữa. Việc đi sâu thêm vào các cấu trúc dữ liệu phức tạp hơn, chính là sự quan tâm của ta trong giáo trình này.

1.2.2. Đối với các bài toán phi số đi đôi với các cấu trúc dữ liệu mới cũng xuất hiện các phép toán mới tác động trên các cấu trúc ấy: Phép tạo lập hay huỷ bỏ một cấu trúc, phép truy cập (access) vào từng phần tử của cấu trúc, phép bổ sung (insertion) hoặc loại bỏ (deletion) một phần tử trên cấu trúc v.v...

Các phép đó sẽ có những tác dụng khác nhau đối với từng cấu trúc. Có phép hữu hiệu đối với cấu trúc này nhưng lại tỏ ra không hữu hiệu trên cấu trúc khác.

Vì vậy chọn một cấu trúc dữ liệu phải nghĩ ngay tới các phép toán tác động trên cấu trúc ấy. Và ngược lại, nói tới phép toán thì lại phải chú ý tới phép đó được tác động trên cấu trúc nào. Cho nên cũng không có gì lạ khi người ta quan niệm: nói tới cấu trúc dữ liệu là bao hàm luôn cả phép toán tác động trên các cấu trúc ấy. Ở giáo trình này tuy ta tách riêng hai khái niệm đó nhưng cấu trúc dữ liệu và các phép toán tương ứng vẫn luôn được trình bày cùng với nhau.

1.2.3. Cách biểu diễn một cấu trúc dữ liệu trong bộ nhớ được gọi là *cấu trúc lưu trữ* (storage structures). Đó chính là cách cài đặt cấu trúc ấy trên máy tính điện tử và trên cơ sở các cấu trúc lưu trữ này mà thực hiện các phép xử lý. Sự phân biệt giữa cấu trúc dữ liệu và cấu trúc lưu trữ tương ứng, cần phải được đặt ra. Có thể có nhiều cấu trúc lưu trữ khác nhau cho cùng một cấu trúc dữ liệu, cũng như có thể có những cấu trúc dữ liệu khác nhau mà được thể hiện trong bộ nhớ bởi cùng một kiểu cấu trúc lưu trữ. Thường khi xử lý, mọi chú ý đều hướng tới cấu trúc lưu trữ, nên ta dễ quên mất cấu trúc dữ liệu tương ứng.

Khi đề cập tới cấu trúc lưu trữ, ta cũng cần phân biệt: cấu trúc lưu trữ tương ứng với bộ nhớ trong - *lưu trữ trong*, hay ứng với bộ nhớ ngoài - *lưu trữ ngoài*. Chúng đều có những đặc điểm riêng và kéo theo các cách xử lý khác nhau.

1.2.4. Thường trong một ngôn ngữ lập trình bao giờ cũng có các cấu trúc dữ liệu tiền định (predefined data structures). Chẳng hạn: cấu trúc *mảng* (array) là cấu trúc rất phổ biến trong các ngôn ngữ. Nó thường được sử dụng để tổ chức các tập dữ liệu, có số lượng ấn định và có cùng kiểu Nếu như sử dụng một ngôn ngữ mà cấu trúc dữ liệu tiền định của nó phù hợp với cấu trúc dữ liệu xác định bởi người dùng thì tất nhiên rất thuận tiện. Nhưng không phải các cấu trúc dữ liệu tiền định của ngôn ngữ lập trình được sử dụng đều đáp ứng được mọi yêu cầu cần thiết về cấu trúc, chẳng hạn nếu xử lý hồ sơ cán bộ mà dùng ngôn ngữ PASCAL, thì ta có thể tổ chức mỗi hồ sơ dưới dạng một *bản ghi* (record) bao gồm nhiều thành phần, mỗi thành phần của bản ghi đó ta gọi là *trường* (field) sẽ không nhất thiết phải cùng kiểu. Ví dụ, trường: "Họ và tên" có *kiểu ký tự* (char), trường: "ngày sinh" có *kiểu số nguyên* (integer).

Nhưng nếu dùng ngôn ngữ FORTRAN thì lại gặp khó khăn. Ta chỉ có thể mô phỏng các mục của hồ sơ dưới dạng các vectơ hay ma trận và do đó việc xử lý sẽ phức tạp hơn.

Cho nên chấp nhận một ngôn ngữ tức là chấp nhận các cấu trúc tiền định của ngôn ngữ ấy và phải biết linh hoạt vận dụng chúng để mô phỏng các cấu trúc dữ liệu đã chọn cho bài toán cần giải quyết.

Tuy nhiên, trong thực tế việc lựa chọn một ngôn ngữ không phải chỉ xuất phát từ yêu cầu của bài toán mà còn phụ thuộc vào rất nhiều yếu tố khách quan cũng như chủ quan của người lập trình nữa.

Tóm lại, trừ vấn đề thứ tư vừa nêu, có thể tách ra và xét riêng, tới đây ta cũng thấy được: ba vấn đề trước đều liên quan tới cấu trúc dữ liệu. Chúng ảnh hưởng trực tiếp đến giải thuật để giải bài toán.

Vì vậy ba vấn đề này chính là đối tượng bàn luận đến trong giáo trình của chúng ta.

1.3 Ngôn ngữ diễn đạt giải thuật

Mặc dầu vấn đề ngôn ngữ lập trình không được đặt ra ở giáo trình này, nhưng để diễn đạt các giải thuật mà ta sẽ trình bày trong giáo trình, ta cũng không thể không lựa chọn một ngôn ngữ. Có thể nghĩ ngay tới việc sử dụng một ngôn ngữ cấp cao hiện có, chẳng hạn PASCAL, C, C++,... nhưng như vậy ta sẽ gặp một số hạn chế sau:

- Phải luôn luôn tuân thủ các quy tắc chặt chẽ về cú pháp của ngôn ngữ đó khiến cho việc trình bày về giải thuật và cấu trúc dữ liệu có thiên hướng nặng nề, gò bó.

- Phải phụ thuộc vào cấu trúc dữ liệu tiền định của ngôn ngữ nên có lúc không thể hiện được đầy đủ các ý về cấu trúc mà ta muốn biểu đạt.

- Ngôn ngữ nào được chọn cũng không hẳn đã được mọi người yêu thích và muốn sử dụng.

Vì vậy, ở đây ta sẽ dùng một ngôn ngữ "thô hơn", có đủ khả năng diễn đạt được giải thuật trên các cấu trúc đề cập đến (mà ta giới thiệu bằng tiếng Việt), với một mức độ linh hoạt nhất định, không quá gò bó, không câu nệ nhiều về cú pháp nhưng cũng gần gũi với các ngôn ngữ chuẩn để khi cần thiết dễ dàng chuyển đổi. Ta tạm gọi nó bằng cái tên: "ngôn ngữ tựa PASCAL". Sau đây là một số quy tắc bước đầu, ở các chương sau sẽ có thể bổ sung thêm.

1.3.1 Quy cách về cấu trúc chương trình

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

Ví dụ:

Program NHAN-MA-TRAN

Độ dài tên không hạn chế.

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

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

1.3.2 Ký tự và biểu thức

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

26 chữ cái Latin in hoa hoặc in thường

10 chữ số thập phân

– Các dấu phép toán số học +, -, *, / , ↑ (luỹ thừa)

– Các dấu phép toán quan hệ <, =, >, ≤, ≥, ≠

– Giá trị logic: **true, false**

– Dấu phép toán logic: **and, or, not**

– Tên biến: dãy chữ cái và chữ số, bắt đầu bằng chữ cái.

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

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

1.3.3 Các câu lệnh (hay các chỉ thị)

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

1.3.3.1 Câu lệnh gán

Có dạng V: = E

với V chỉ tên biến, tên hàm

E chỉ biểu thức

Ở đây cho phép dùng phép gán chung.

Ví dụ:

A:= B := 0.1

1.3.3.2 Câu lệnh ghép

Có dạng:

Begin S_1, S_2,...; S_n; **end**

với S_i, i = 1, ..., n là các câu lệnh.

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

1.3.3.3 Câu lệnh điều kiện

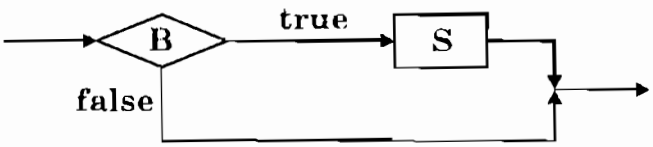
Có dạng:

if B **then** S

với B là biểu thức logic

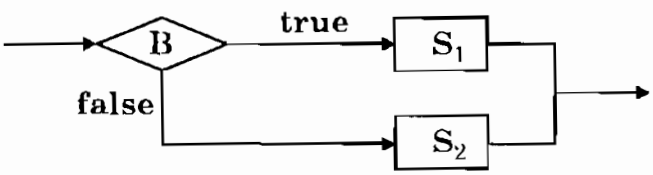
S là một câu lệnh khác

có thể diễn tả bởi sơ đồ



hoặc:

if B **then** S_1 **else** S_2



1.3.3.4 Câu lệnh tuyển

Case

B_1: S_1;

B_2: S_2;

...

...

B_n:S_n;

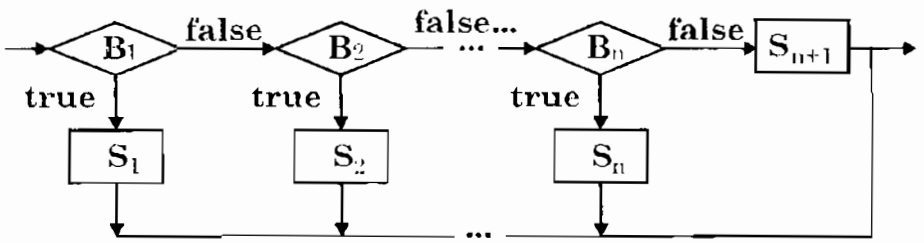
else: S_{n+1}

end case

với: B_i (i = 1, 2, ..., n) là các điều kiện

S_i (i = 1, 2, ..., n) là các câu lệnh

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



* Vài điểm linh động

else có thể không có mặt.

S_i (i = 1, 2, ..., n) có thể được thay bằng một dãy các câu lệnh thể hiện một dãy xử lý khi có điều kiện B_i mà không cần phải đặt giữa **begin** và **end**

1.3.3.5 Câu lệnh lặp

* Với số lần lặp biết trước

for i := m **to** n **do** S

nhằm thực hiện câu lệnh S với i lấy giá trị nguyên từ m tới n (n ≥m), với bước nhảy tăng bằng 1;

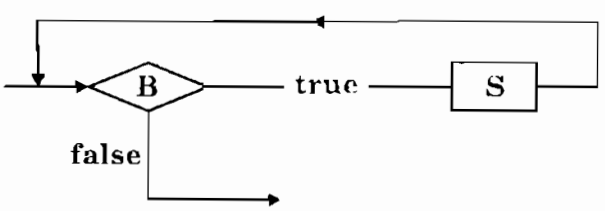
Hoặc:

for i := n **down to** m **do** S

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

* Với số lần lặp không biết trước

while B **do** S

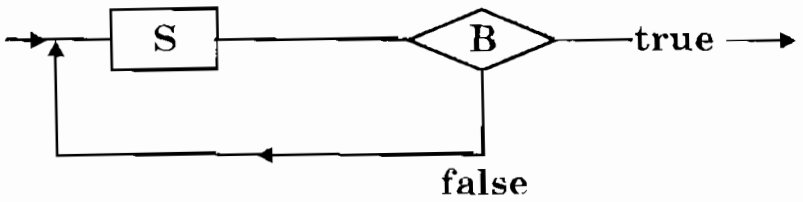


chừng nào mà B có giá trị bằng true thì thực hiện S,

hoặc:

repeat S **until** B

Lặp lại S cho tới khi B có giá trị true (S có thể là một dãy lệnh)



1.3.3.6 Câu lệnh chuyển

go to n (n là số hiệu của một bước trong chương trình)

Thường người ta hạn chế việc dùng **go to**. Tuy nhiên với mục đích diễn đạt cho tự nhiên, trong một chừng mực nào đó ta vẫn sử dụng.

1.3.3.7 Câu lệnh vào, ra

Có dạng:

read (< danh sách biến>)

write (<danh sách biến hoặc dòng ký tự>)

Các biến trong danh sách cách nhau bởi dấu phẩy.

Dòng ký tự là một dãy các ký tự đặt giữa hai dấu nháy ' '.

1.3.3.8 Câu lệnh kết thúc chương trình

end

1.3.4 Chương trình con

* Chương trình con hàm

Có dạng:

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

$S_1, S_2, ...; S_n$

return

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

* Chương trình con thủ tục

Tương tự như trên, chỉ khác ở chỗ:

Từ khoá **procedure** thay cho **function.**

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

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

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

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

Như vậy nghĩa là các chương trình được nêu ra chỉ là đoạn thể hiện các phép xử lý theo giải thuật đã định, trên các cấu trúc dữ liệu được mô tả trước đó, bằng ngôn ngữ tự nhiên.

BÀI TẬP CHƯƠNG 1

1.1. Tìm thêm các ví dụ minh họa mối quan hệ giữa cấu trúc dữ liệu và giải thuật.

1.2. Các bài toán phi số khác với các bài toán khoa học kỹ thuật ở những đặc điểm gì?

1.3. Cấu trúc dữ liệu và cấu trúc lưu trữ khác nhau ở chỗ nào?

1.4. Hãy nêu một vài cấu trúc dữ liệu tiền định của các ngôn ngữ mà anh (chị) biết.

1.5. Các cấu trúc dữ liệu tiền định trong một ngôn ngữ có đủ đáp ứng mọi yêu cầu về tổ chức dữ liệu không?

Có thể có cấu trúc dữ liệu do người dùng định ra không?

1.6. Một chương trình PASCAL có phải là một tập dữ liệu có cấu trúc không?

1.7. Hãy nêu các tính chất của một giải thuật và cho ví dụ minh hoạ.

Chương 2

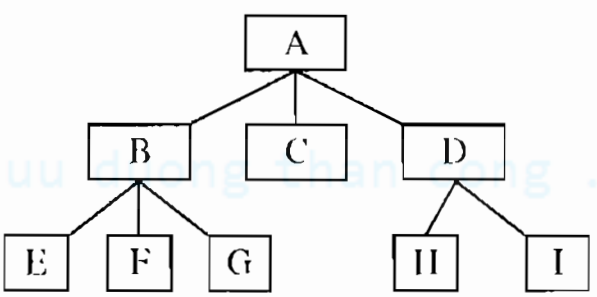
THIẾT KẾ VÀ PHÂN TÍCH GIẢI THUẬT

2.1 Từ bài toán đến chương trình

2.1.1 Mô-đun hoá và việc giải quyết bài toán

Các bài toán giải được trên máy tính điện tử ngày càng đa dạng và phức tạp. Các giải thuật và chương trình để giải chúng cũng ngày càng có quy mô lớn và càng khó khi thiết lập cũng như khi muốn tìm hiểu.

Tuy nhiên, ta cũng thấy rằng mọi việc sẽ đơn giản hơn nếu như có thể phân chia bài toán lớn của ta thành các bài toán nhỏ. Điều đó cũng có nghĩa là nếu coi bài toán của ta như một mô-đun chính thì cần chia nó thành các mô-đun con, và dĩ nhiên, với tinh thần như thế, đến lượt nó, mỗi mô-đun này lại được phân chia tiếp cho tới những mô-đun ứng với các phần việc cơ bản mà ta đã biết cách giải quyết. Như vậy việc tổ chức lời giải của bài toán sẽ được thể hiện theo một cấu trúc phân cấp, có dạng như hình sau:



Hình 2.1

Chiến thuật giải quyết bài toán theo tinh thần như vậy chính là chiến thuật "chia để trị" (divide and conquer). Để thể hiện chiến thuật đó, người ta dùng cách thiết kế *"từ đỉnh xuống"* (top-down design). Đó là cách phân

tích tổng quát toàn bộ vấn đề, xuất phát từ dữ kiện và các mục tiêu đặt ra, để để cập đến những công việc chủ yếu, rồi sau đó mới đi dần vào giải quyết các phần cụ thể một cách chi tiết hơn (cũng vì vậy mà người ta gọi là cách *thiết kế từ khái quát đến chi tiết*). Ví dụ ta nhận được từ Chủ tịch Hội đồng xét cấp học bổng của trường một yêu cầu là:

"Dùng máy tính điện tử để quản lý và bảo trì các hồ sơ về học bổng của các sinh viên ở diện được tài trợ, đồng thời thường kỳ phải lập các báo cáo tổng kết để đệ trình lên Bộ".

Như vậy trước hết ta phải hình dung được cụ thể hơn đầu vào và đầu ra của bài toán.

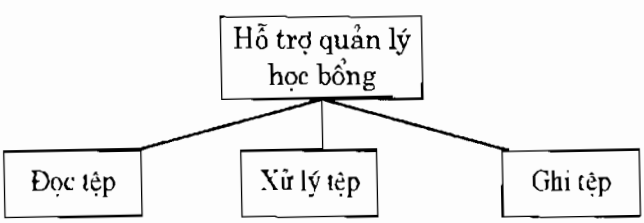
Có thể coi như ta đã có một tập các hồ sơ (mà ta gọi là *tệp* - file) bao gồm các bản ghi (records) về các thông tin liên quan tới học bổng của sinh viên, chẳng hạn: số hiệu sinh viên, điểm trung bình (theo học kỳ), điểm đạo đức, khoản tiền tài trợ Và chương trình lập ra phải tạo điều kiện cho người sử dụng giải quyết được các yêu cầu sau:

1) Tìm lại và hiển thị được bản ghi của bất kỳ sinh viên nào tại thiết bị cuối (terminal) của người dùng.
2) Cập nhật (update) được bản ghi của một sinh viên cho trước bằng cách thay đổi điểm trung bình, điểm đạo đức, khoản tiền tài trợ, nếu cần.
3) In bản tổng kết chứa những thông tin hiện thời (đã được cập nhật mỗi khi có thay đổi) gồm số hiệu, điểm trung bình, điểm đạo đức, khoản tiền tài trợ.

Xuất phát từ những nhận định nêu trên, giải thuật xử lý sẽ phải giải quyết ba nhiệm vụ chính như sau:

1) Những thông tin về sinh viên được học bổng, lưu trữ trên đĩa phải được đọc vào bộ nhớ trong để có thể xử lý (ta gọi là nhiệm vụ "đọc tệp").
2) Xử lý các thông tin này để tạo ra kết quả mong muốn (nhiệm vụ: "xử lý tệp").
3) Sao chép những thông tin đã được cập nhật vào tệp trên đĩa để lưu trữ cho việc xử lý sau này (nhiệm vụ: "ghi tệp").

Có thể hình dung, cách thiết kế này theo sơ đồ cấu trúc ở hình 2.2

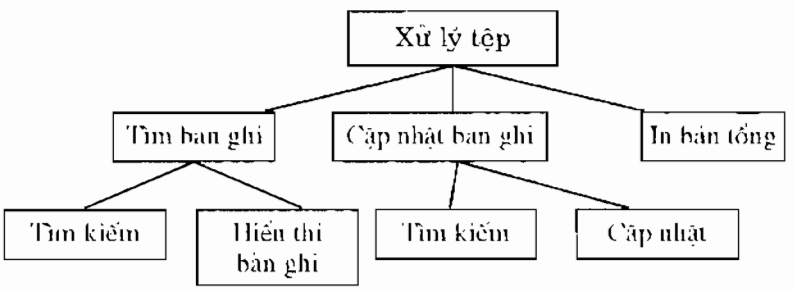


Hình 2.2

Các nhiệm vụ ở mức đầu này thường tương đối phức tạp, cần phải chia thành các nhiệm vụ con. Chẳng hạn, nhiệm vụ "xử lý tệp" sẽ được phân thành ba, tương ứng với việc giải quyết ba yêu cầu chính đã được nêu ở trên:

1. Tìm lại bản ghi của một sinh viên cho trước
2. Cập nhật thông tin trong bản ghi sinh viên
3. In bảng tổng kết những thông tin về các sinh viên được học bổng.

Những nhiệm vụ con này cũng có thể chia thành nhiệm vụ nhỏ hơn. Có thể hình dung theo sơ đồ cấu trúc như sau:



Hình 2.3

Cách thiết kế giải thuật theo kiểu top-down như trên giúp cho việc giải quyết bài toán được định hướng rõ ràng, tránh sa đà ngay vào các chi tiết phụ. Nó cũng là nền tảng cho việc lập trình có cấu trúc.

Thông thường, đối với các bài toán lớn, việc giải quyết nó phải do nhiều người cùng làm. Chính phương pháp mô-đun hoá sẽ cho phép tách bài toán ra thành các phần độc lập tạo điều kiện cho các nhóm giải quyết phần việc của mình mà không làm ảnh hưởng gì đến nhóm khác. Với chương trình được xây dựng trên cơ sở của các giải thuật được thiết kế theo cách này thì việc tìm hiểu cũng như sửa chữa chỉnh lý sẽ dễ dàng hơn.

Việc phân bài toán thành các bài toán con như thế không phải là một việc làm dễ dàng. Chính vì vậy mà có những bài toán nhiệm vụ phân tích và thiết kế giải thuật giải bài toán đó còn mất nhiều thời gian và công sức hơn cả nhiệm vụ lập trình.

2.1.2 Phương pháp tinh chỉnh từng bước (Stepwise refinement)

Tinh chỉnh từng bước là phương pháp thiết kế giải thuật gắn liền với lập trình. Nó phản ánh tinh thần của quá trình mô-đun hoá bài toán và thiết kế kiểu top-down.