ỦY BAN NHÂN DÂN TỈNH AN GIANG
TRƯỜNG CAO ĐẲNG NGHỀ AN GIANG

GIÁO TRÌNH

Cấu trúc dữ liệu và giải thuật

NGHỀ LẬP TRÌNH MÁY TÍNH & TIN ỨNG DỤNG

TRÌNH ĐỘ CAO ĐẲNG NGHỀ & TRUNG CẤP NGHỀ

(Ban hành theo Quyết định số: /QĐ-CĐN ngày tháng năm 20 của Hiệu trưởng trường Cao đẳng nghề An Giang)

Tên tác giả : Trần Thị Kim Ngọc
Năm ban hành: 2018

TUYÊN BỐ BẢN QUYỀN

LỜI GIỚI THIỆU

Bài giảng cấu trúc dữ liệu và giải thuật được viết nhằm để giảng dạy cho sinh viên chuyên ngành CNTT trường Cao Đẳng Nghề An giang. Bài giảng được thiết kế theo chương trình môn học cấu trúc dữ liệu và giải thuật của Bộ ban hành. Bài giảng này bao gồm 4 chương, nội dung các chương được trình bày các cấu trúc dữ liệu và các giải thuật đơn giản nhất của tin học. Trước tiên bài giảng trình bày về các cấu trúc dữ liệu cơ bản như: mảng, con trỏ, cấu trúc, tập tin; tiếp theo là các cấu trúc dữ liệu nâng cao như: danh sách, ngăn xếp, hàng đợi và một số ứng dụng của danh sách. Sau đó bài giảng trình bày tiếp về giải thuật sắp xếp và tìm kiếm, với các phương pháp sắp xếp: xen, chọn, nổi bọt, quick sort và tìm kiếm như: tuần tự và nhị phân. Thêm vào đó, cuối chương sẽ có các bài tập tương ứng để sinh viên có thể ôn lại lý thuyết và tùy vào mỗi chương mà có một số bài tập nâng cao để khuyến khích sinh viên tự học và nghiên cứu.

Cuốn tài liệu giảng dạy này vẫn còn nhiều thiếu sót và hạn chế. Rất mong nhận được ý kiến đóng góp của sinh viên và các bạn đọc để bài giảng ngày càng hoàn thiện hơn.

Chân thành cảm ơn quý Thầy Cô trong Hội đồng thẩm định của trường Cao Đẳng Nghề An Giang để bài giảng Cấu trúc dữ liệu và giải thuật được hoàn chỉnh.

An Giang, ngày tháng năm 2018

Tham gia biên soạn

Trần Thị Kim Ngọc

MỤC LỤC

GIÁO TRÌNH MÔN HỌC CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Tên môn học: CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Mã môn học: MH 16

Thời gian thực hiện môn học: 60 giờ (Lý thuyết: 20 giờ, thực hành, thí nghiệm, thảo luận: 36 giờ, kiểm tra: 4 giờ).

Vị trí, tính chất, ý nghĩa và vai trò của môn học:

- Vị trí: Thuộc nhóm môn: Môn học lý thuyết chuyên nghành, được bố trí sau các môn: Tin học căn bản, Lập trình căn bản

- Tính chất: Môn học này yêu cầu phải có tư duy logic và các kiến thức về lập trình căn bản.

-Ý nghĩa và vai trò của môn học: giúp các em tư duy, vận động suy nghĩ làm nền tảng cho các môn học sau.

Mục tiêu của môn học**:**

- Về kiến thức

- Hiểu được nội dung của: dữ liệu, giải thuật, mối quan hệ giữa cấu trúc dữ liệu và giải thuật.
- Phân tích và xác định được dữ liệu, giải thuật, sự kết hợp của dữ liệu và giải thuật trong một chương trình máy tính.
- Áp dụng thuật toán hợp lý đối với cấu trúc dữ liệu tương thích để giải quyết bài toán đơn giản.
- Áp dụng được các phương pháp sắp xếp, tìm kiếm đơn giản, các cấu trúc dữ liệu động (danh sách liên kết) vào giải quyết các bài toán.

- Về kỹ năng: Thực hành (cài đặt và biên dịch) các bài toán sử dụng các cấu trúc và giải thuật đã học.

- Về năng lực tự chủ và trách nhiệm:

- Nghiêm túc trong học tập và thực hiện tốt các yêu cầu được giao.
- Luôn động não suy nghĩ. thường xuyên luyện tập tư duy trong việc học
- Rèn luyện tính cẩn thận, khoa học trong công việc học tập, nghiên cứu.

CHƯƠNG 1: GIỚI THIỆU CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Mục tiêu:

Nhằm cung cấp những kiến thức cơ bản về cấu trúc dữ liệu, giải thuật, kiểu dữ liệu, mô hình dữ liệu và các kiến thức về thiết kế, phân tích giải thuật cũng như các phương pháp phân tích, thiết kế giải thuật.

Nội dung chính:

I. MỐI LIÊN HỆ GIẢI THUẬT VÀ CẤU TRÚC DỮ LIỆU

Theo quan điểm của phân tích thiết kế hướng đối tượng, mỗi lớp sẽ được xây dựng với một số chức năng nào đó và các đối tượng của nó sẽ tham gia vào hoạt động của chương trình. Điểm mạnh của hướng đối tượng là tính đóng kín và tính sử dụng lại của các lớp. Mỗi phần mềm biên dịch cho một ngôn ngữ lập trình nào đó đều chứa rất nhiều thư viện các lớp như vậy. Chúng ta thử điểm qua một số lớp mà người lập trình thường hay sử dụng: các lớp có nhiệm vụ đọc/ ghi để trao đổi dữ liệu với các thiết bị ngoại vi như đĩa, máy in, bàn phím,…; các lớp đồ họa cung cấp các chức năng vẽ, tô màu cơ bản; các lớp điều khiển cho phép xử lý việc giao tiếp với người sử dụng thông qua bàn phím, chuột, màn hình; các lớp phục vụ các giao dịch truyền nhận thông tin qua mạng;…Các lớp cấu trúc dữ liệu mà chúng ta sắp bàn đến cũng không là một trường hợp ngoại lệ. Có thể chia tất cả các lớp này thành hai nhóm chính:

- Các lớp dịch vụ.

- Các lớp có khả năng lưu trữ và xử lý lượng dữ liệu lớn.

Nhóm thứ hai muốn nói đến các lớp cấu trúc dữ liệu. Vậy có gì giống và khác nhau giữa các lớp cấu trúc dữ liệu và các lớp khác?

- Điểm giống nhau giữa các lớp cấu trúc dữ liệu và các lớp khác: Mỗi lớp đều phải thực hiện một số chức năng thông qua các hành vi của các đối tượng của nó. Một khi chúng ta đã xây dựng xong một lớp cấu trúc dữ liệu nào đó, chúng ta hoàn toàn tin tưởng rằng nó sẽ hoàn thành xuất sắc những nhiệm vụ mà chúng ta đã thiết kế và đã giao phó cho nó. Điều này rất khác biệt so với những tài liệu viết về cấu trúc dữ liệu theo quan điểm hướng thủ tục trước đây: việc xử lý dữ liệu không hề có tính đóng kín và tính sử dụng lại. Tuy về mặt thực thi thì các chương trình như thế có khả năng chạy nhanh hơn, nhưng chúng bộc lộ rất nhiều nhược điểm: thời gian phát triển giải thuật chính rất chậm gây khó khăn nhiều cho người lập trình, chương trình thiếu tính trong sáng, rất khó sửa lỗi và phát triển.

- Đặc trưng riêng của các lớp cấu trúc dữ liệu: Nhiệm vụ chính của các lớp cấu trúc dữ liệu là nắm giữ dữ liệu sao cho có thể đáp ứng mỗi khi được chương trình yêu cầu trả về một dữ liệu cụ thể nào đó mà chương trình cần đến. Những thao tác cơ bản đối với một cấu trúc dữ liệu thường là: thêm dữ liệu mới, xóa bỏ dữ liệu đã

có, tìm kiếm, truy xuất.

Ngoài các thao tác dữ liệu cơ bản, các cấu trúc dữ liệu khác nhau sẽ khác nhau về các thao tác bổ sung khác. Chính vì điều này mà khi thiết kế những giải thuật để giải quyết các bài toán lớn, người ta sẽ lựa chọn cấu trúc dữ liệu nào là thích hợp nhất.

Chúng ta thử xem xét một ví dụ thật đơn giản sau đây:

Giả sử chúng ta cần viết một chương trình nhận vào một dãy các con số, và in chúng ra theo thứ tự ngược với thứ tự nhập vào ban đầu.

Để giải quyết bài toán này, nếu chúng ta nghĩ đến việc phải khai báo các biến để lưu các giá trị nhập vào như thế nào, và sau đó là thứ tự in ra sao để đáp ứng yêu cầu bài toán, thì dường như là chúng ta đã quên áp dụng nguyên tắc lập trình hướng đối tượng: chúng ta đã phải bận tâm đến những việc quá chi tiết. Đây chỉ là một ví dụ vô cùng đơn giản, nhưng nó có thể minh họa cho vai trò của cấu trúc dữ liệu.

Nếu chúng ta nhớ rằng, việc tổ chức và lưu dữ liệu như thế nào là một việc quá chi tiết và tỉ mỉ không nên thực hiện vào lúc này, thì đó chính là lúc chúng ta đã bước đầu hiểu được vai trò của các lớp cấu trúc dữ liệu.

Môn cấu trúc dữ liệu và giải thuật sẽ giúp chúng ta hiểu rõ về các lớp cấu trúc dữ liệu có sẵn trong các phần mềm. Hơn thế nữa, trong khi học cách xây dựng các lớp cấu trúc dữ liệu từ đơn giản đến phức tạp, chúng ta sẽ nắm được các phương pháp cũng như các kỹ năng thông qua một số nguyên tắc chung. Từ đó, ngoài khả năng hiểu rõ để có thể lựa chọn một cách đúng đắn nhất những cấu trúc dữ liệu có sẵn, chúng ta còn có khả năng xây dựng những lớp cấu trúc dữ liệu phức tạp hơn, tinh tế và thích hợp hơn trong mỗi bài toán mà chúng ta cần giải quyết. Khả năng thừa kế các cấu trúc dữ liệu có sẵn để phát triển thêm các tính năng mong muốn cũng là một điều đáng lưu ý.

Với ví dụ trên, những ai đã từng tiếp xúc ít nhiều với việc lập trình đều không xa lạ với khái niệm "ngăn xếp". Đây là một cấu trúc dữ liệu đơn giản nhất nhưng lại rất thông dụng, và dĩ nhiên chúng ta sẽ có dịp học kỹ hơn về nó. Ở đây chúng ta muốn mượn nó để minh họa, và cũng nhằm giúp cho người đọc làm quen với một phương pháp tiếp cận hoàn toàn nhất quán trong suốt giáo trình này.

Giả sử cấu trúc dữ liệu ngăn xếp của chúng ta đã được giao cho một nhiệm vụ là cất giữ những dữ liệu và trả về khi có yêu cầu, theo một quy định bất di bất dịch là dữ liệu đưa vào sau phải được lấy ra trước. Bằng cách sử dụng cấu trúc dữ liệu ngăn xếp, chương trình trở nên hết sức đơn giản và được trình bày bằng ngôn ngữ giả như sau:

Lặp cho đến khi nhập đủ các con số mong muốn

{

Nhập 1 con số.

Cất vào ngăn xếp con số vừa nhập.

}

Lặp trong khi mà ngăn xếp vẫn còn dữ liệu

{

Lấy từ ngăn xếp ra một con số. In số vừa lấy được.

}

Chúng ta sẽ có dịp gặp nhiều bài toán phức tạp hơn mà cũng cần sử dụng đến đặc tính này của ngăn xếp. Tính đóng kín của các lớp giúp cho chương trình vô cùng trong sáng. Đoạn chương trình trên không hề cho chúng ta thấy ngăn xếp đã làm việc với các dữ liệu được đưa vào như thế nào, đó là nhiệm vụ mà chúng ta đã giao phó cho nó và chúng ta hoàn toàn yên tâm về điều này. Bằng cách này, khi đã có những cấu trúc dữ liệu thích hợp, người lập trình có thể dễ dàng giải quyết các bài toán lớn. Họ có thể yên tâm tập trung vào những điểm mấu chốt để xây dựng, tinh chế giải thuật và kiểm lỗi.

Trên đây chúng ta chỉ vừa mới giới thiệu về phần cấu trúc dữ liệu nằm trong nội dung của môn học "cấu trúc dữ liệu và giải thuật". Vậy giải thuật là gì? Đứng trên quan điểm thiết kế và lập trình hướng đối tượng, chúng ta đã hiểu vai trò của các lớp. Vậy khi đã có các lớp rồi thì người ta cần xây dựng kịch bản cho các đối tượng hoạt động nhằm giải quyết bài toán chính. Chúng ta cần một cấu trúc chương trình để tạo ra kịch bản đó: việc gì làm trước, việc gì làm sau; việc gì chỉ làm trong những tình huống đặc biệt nào đó; việc gì cần làm lặp lại nhiều lần. Chúng ta nhắc đến giải thuật chính là quay về với khái niệm của "lập trình thủ tục" trước kia. Ngoài ra, chúng ta cũng cần đến giải thuật khi cần hiện thực cho mỗi lớp: xong phần đặc tả các phương thức - phương tiện giao tiếp của lớp với bên ngoài - chúng ta cần đến khái niệm "lập trình thủ tục" để giải quyết phần hiện thực bên trong của các phương thức này. Đó là việc chúng ta phải xử lý những dữ liệu bên trong của chúng như thế nào mới có thể hoàn thành được chức năng mà phương thức phải đảm nhiệm.

Như vậy, về phần giải thuật trong môn học này, chủ yếu chúng ta sẽ tìm hiểu các giải thuật mà các phương thức của các lớp Cấu trúc dữ liệu dùng đến, một số giải thuật sắp xếp tìm kiếm, và các giải thuật trong các ứng dụng minh họa việc sử dụng các lớp cấu trúc dữ liệu để giải quyết một số bài toán đó.

II. ĐÁNH GIÁ ĐỘ PHỨC TẠP CỦA GIẢI THUẬT

1. Sự cần thiết phải phân tích giải thuật:

Trong khi giải một bài toán chúng ta có thể có một số giải thuật khác nhau, vấn đề là cần phải đánh giá các giải thuật đó để lựa chọn một giải thuật tốt (nhất). Thông thường thì ta sẽ căn cứ vào các tiêu chuẩn sau:

(1) Giải thuật đúng đắn

(2) Giải thuật đơn giản

(3) Giải thuật thực hiện nhanh

Với yêu cầu (1), để kiểm tra tính đúng đắn của giải thuật chúng ta có thể cài đặt giải thuật đó và cho thực hiện trên máy với một số bộ dữ liệu mẫu rồi lấy kết quả thu được so sánh với kết quả đã biết. Thực ra thì cách làm này không chắc chắn bởi vì có thể giải thuật đúng với tất cả các bộ dữ liệu chúng ta đã thử nhưng lại sai với một bộ dữ liệu nào đó. Vả lại cách làm này chỉ phát hiện ra giải thuật sai chứ chưa chứng minh được là nó đúng. Tính đúng đắn của giải thuật cần phải được chứng minh bằng toán học. Tất nhiên điều này không đơn giản và do vậy chúng ta sẽ không đề cập đến ở đây.

Khi chúng ta viết một chương trình để sử dụng một vài lần thì yêu cầu (2) là quan trọng nhất. Chúng ta cần một giải thuật dễ viết chương trình để nhanh chóng có được kết quả, thời gian thực hiện chương trình không được đề cao vì dù sao thì chương trình đó cũng chỉ sử dụng một vài lần mà thôi.

Tuy nhiên khi một chương trình được sử dụng nhiều lần thì yêu cầu tiết kiệm thời gian thực hiện chương trình lại rất quan trọng đặc biệt đối với những chương trình mà khi thực hiện cần dữ liệu nhập lớn do đó yêu cầu (3) sẽ được xem xét một cách kỹ càng. Ta gọi nó là hiệu quả thời gian thực hiện giải thuật.

2. Thời gian thực hiện của giải thuật:

Một phương pháp hiệu quả để xác định hiệu quả thời gian thực hiện của một giải thuật là lập trình nó và đo lường thời gian thực hiện của hoạt động trên một máy tính xác định đối với tập hợp được chọn lọc các dữ liệu vào.

Thời gian thực hiện không chỉ phụ thuộc vào giải thuật mà còn phụ thuộc vào tập các chỉ thị của máy tính, chất lượng của máy tính và kỹ xảo của người lập trình. Sự thi hành cũng có thể điều chỉnh để thực hiện tốt trên tập đặc biệt các dữ liệu vào được chọn. Để vượt qua các trở ngại này, các nhà khoa học máy tính đã chấp nhận độ phức tạp của thời gian được tiếp cận như một sự đo lường cơ bản sự thực thi của giải thuật. Thuật ngữ tính hiệu quả sẽ đề cập đến sự đo lường này và đặc biệt đối với độ phức tạp thời gian trong trường hợp xấu nhất.

a. Thời gian thực hiện chương trình:

Thời gian thực hiện một chương trình là một hàm của kích thước dữ liệu vào, ký hiệu T(n), trong đó n là kích thước (độ lớn) của dữ liệu vào.

Ví dụ:

Chương trình tính tổng của n số có thời gian thực hiện là T(n) = cn trong đó c là một hằng số.

Thời gian thực hiện chương trình là một hàm không âm, tức là $T(n) \geq 0 \ \forall \ n \geq 0$.

b. Đơn vị đo thời gian thực hiện

Đơn vị của T(n) không phải là đơn vị đo thời gian bình thường như giờ, phút giây … mà thường được xác định bởi số các lệnh được thực hiện trong một máy tính lý tưởng.

Ví dụ:

Khi ta nói thời gian thực hiện của một chương trình là T(n) = cn thì có nghĩa là chương trình ấy cần cn chỉ thị thực thi.

c. Thời gian thực hiện trong trường hợp xấu nhất:

Nói chung thì thời gian thực hiện chương trình không chỉ phụ thuộc vào kích thước mà còn phụ thuộc vào tính chất của dữ liệu vào. Nghĩa là dữ liệu vào có cùng kích thước nhưng thời gian thực hiện chương trình có thể khác nhau. Chẳng hạn chương trình sắp xếp dãy số nguyên tăng dần, khi ta cho vào dãy có thứ tự thì thời gian thực hiện khác với khi ta cho vào dãy chưa có thứ tự, hoặc khi ta cho vào một dãy đã có thứ tự tăng thì thời gian thực hiện cũng khác so với khi ta cho vào một dãy đã có thứ tự giảm

Vì vậy thường ta coi T(n) là thời gian thực hiện chương trình trong trường hợp xấu nhất trên dữ liệu vào có kích thước n, tức là: T(n) là thời gian lớn nhất để thực hiện chương trình đối với mọi dữ liệu vào có cùng kích thước n.

3. Tỷ suất tăng và độ phức tạp của giải thuật

a. Tỷ suất tăng:

Ta nói rằng hàm không âm T(n) có tỷ suất tăng (growth rate) f(n) nếu tồn tại các hằng số c và n_o sao cho $T(n) \leq cf(n)$ với mọi $n \geq n_o$

Ta có thể chứng minh được rằng " Cho một hàm không âm T(n) bất kỳ, ta luôn tìm được tỷ suất tăng f(n) của nó"

Ví dụ:

Giả sử T(0) = 1, T(1) = 4 và tổng quát $T(n) = (n+1)^2$. Đặt $n_o = 1$ và c = 4 thì với mọi $n \geq 1$ chúng ta dễ dàng chứng minh rằng $T(n) = (n+1)^2 \leq 4n^2$ với mọi $n \geq 1$, tức là tỷ suất tăng của T(n) là n^2

Ví dụ:

Tỷ suất tăng của hàm $T(n) = 3n^3 + 2n^2$ là n^3. Thực vậy, cho no = 0 và c = 5 ta dễ dàng chứng minh rằng với mọi $n \geq 0$ thì $3n^3 + 2n^2 \leq 5n^3$

b. Khái niệm độ phức tạp của giải thuật

Giả sử ta có hai giải thuật P1 và P2 với thời gian thực hiện tương ứng là T1(n) = $100n^2$ (với tỷ suất là n^2) và T2(n) = $5n^3$ (với tỷ suất tăng là n^3). Giải thuật nào sẽ thực hiện nhanh hơn? Câu trả lời phụ thuộc vào kích thước dữ liệu vào. Với n < 20 thì P2 sẽ nhanh hơn P1 (T2 < T1), do hệ số của $5n^3$ nhỏ hơn hệ số của $100n^2$ (5<100). Nhưng khi n>20 thì ngược lại do số mũ của $100n^2$ nhỏ hơn số mũ của $5n^3$ (2 < 3). Ở đây chúng ta chỉ nên quan tâm đến trường hợp n > 20 vì khi n < 20 thì thời gian thực hiện của cả P1 và P2 đều không lớn và sự khác biệt giữa T1 và T2 là không đáng kể.

Như vậy một cách hợp lý là ta xét tỷ suất tăng của hàm thời gian thực hiện chương trình thay vì xét chính bản thân thời gian thực hiện

Cho một hàm T(n), T(n) gọi là có độ phức tạp f(n) nếu tồn tại các hằng c, No sao cho T(n) ≤ cf(n) với mọi n ≥ No (tức là T(n) có tỷ suất tăng là f(n)) và ký hiệu T(n) là O(f(n)) (đọc là "ô của f(n)")

Ví dụ:

T(n) = $(n+1)^2$ có tỷ suất tăng là n^2 nên T(n) = $(n+1)^2$ là O(n^2)

Chú ý: O(c.f(n)) = O(f(n)) với c là hằng số. Đặc biệt O(c) = O(1)

Nói cách khác độ phức tạp tính toán của giải thuật là một hàm chặn trên của hàm thời gian. Vì hằng nhân tử c trong hàm chặn trên không có ý nghĩa nên ta có thể bỏ qua vì vậy hàm thể hiện độ phức tạp có các dạng thường gặp sau: $\log_2 n$, n, $n\log_2 n$, n^2, n^3, 2^n, n!, n^n. Ba hàm cuối cùng ta gọi là dạng hàm mũ, các hàm khác gọi là hàm đa thức. Một giải thuật mà thời gian thực hiện có độ phức tạp là một hàm đa thức thì chấp nhận được tức là có thể cài đặt để thực hiện, còn các giải thuật có độ phức tạp hàm mũ thì phải tìm cách cải tiến giải thuật.

Khi nói đến độ phức tạp của giải thuật là ta muốn nói đến hiệu quả của thời gian thực hiện của chương trình nên ta có thể xem việc xác định thời gian thực hiện của chương trình chính là xác định độ phức tạp của giải thuật.

4. Cách tính độ phức tạp:

Cách tính độ phức tạp của một giải thuật bất kỳ là một vấn đề không đơn giản. Tuy nhiên ta có thể tuân theo một số nguyên tắc sau:

a. Qui tắc cộng:

Nếu T1(n) và T2(n) là thời gian thực hiện của hai đoạn chương trình P1 và P2 và T1(n) = O(f(n)), T2(n) = O(g(n)) thì thời gian thực hiện của đoạn hai chương trình đó nối tiếp nhau là T(n) = O(max(f(n),g(n)))

Ví dụ:

Lệnh gán x:=15 tốn một hằng thời gian hay O(1)

Lệnh đọc dữ liệu READ(x) tốn một hằng thời gian hay O(1)

Vậy thời gian thực hiện cả hai lệnh trên nối tiếp nhau là O(max(1,1)) = O(1)

b. Qui tắc nhân:

Nếu T1(n) và T2(n) là thời gian thực hiện của hai đoạn chương trình P1 và P2 và T1(n) = O(f(n)), T2(n) = O(g(n)) thì thời gian thực hiện của đoạn hai đoạn chương trình đó lồng nhau là T(n) = O(f(n).g(n))

c. Qui tắc tổng quát để phân tích một chương trình:

- Thời gian thực hiện của mỗi lệnh gán, đọc, ghi là O(1)

- Thời gian thực hiện của một chuỗi tuần tự các lệnh được xác định bằng qui tắc cộng. Như vậy thời gian này là thời gian thi hành một lệnh nào đó lâu nhất trong chuỗi lệnh.

- Thời gian thực hiện cấu trúc IF là thời gian lớn nhất thực hiện lệnh sau THEN hoặc sau ELSE và thời gian kiểm tra điều kiện. Thường thời gian kiểm tra điều kiện là O(1)

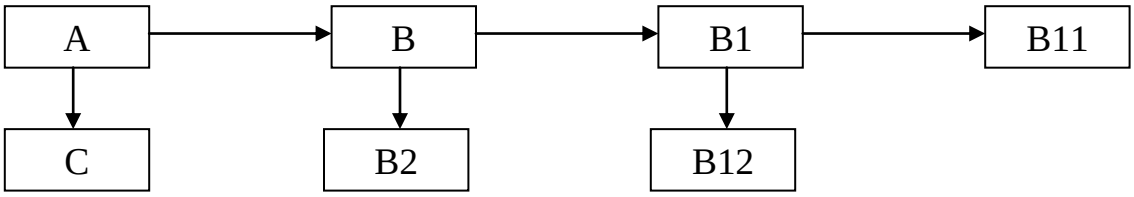
- Thời gian thực hiện vòng lặp là tổng (trên tất cả các lần lặp) thời gian thực hiện thân vòng lặp. Nếu thời gian thực hiện thân vòng lặp không đổi thì thời gian thực hiện vòng lặp là tích của số lần lặp với thời gian thực hiện thân vòng lặp

d. Độ phức tạp của chương trình có gọi chương trình con không đệ quy:

Nếu chúng ta có một chương trình với các chương trình con không đệ quy, để tính thời gian thực hiện của chương trình, trước hết chúng ta tính thời gian thực hiện của các chương trình con không gọi các chương trình con khác. Sau đó chúng ta tính thời gian thực hiện của các chương trình con chỉ gọi các chương trình con mà thời gian thực hiện của chúng đã được tính. Chúng ta tiếp tục quá trình đánh giá thời gian thực hiện của mỗi chương trình con sau khi thời gian thực hiện của tất cả các chương trình con mà nó gọi đã được đánh giá. Cuối cùng ta tính thời gian cho chương trình chính.

Giả sử ta có một hệ thống các chương trình gọi theo sơ đồ sau:

Chương trình A gọi hai chương trình con là B và C, chương trình B gọi hai chương trình con là B1 và B2, chương trình B1 gọi hai chương trình con là B11 và B12



Sơ đồ gọi thực hiện các chương trình con không đệ quy

Để tính thời gian thực hiện của A ta tính theo các bước sau:

- Tính thời gian thực hiện của C, B2, B11 và B12
- Tính thời gian thực hiện của B1
- Tính thời gian thực hiện của B
- Tính thời gian thực hiện của A

Ví dụ:

```
void BubbleSort(int M[], int N)
{1}      { for (int I = 0; I < N-1; I++)
{2}            for (int J = N-1; J > I; J--)
{3}              if (M[J] < M[J-1])
        Swap(M[J], M[J-1]);
        return;
}
void Swap(int &X, int &Y)
{ int Temp = X;
  X = Y;
  Y = Temp;
  return;
}
```

Trong cách viết trên, chương trình Bubble gọi chương trình con Swap, do đó để tính thời gian thực hiện của bubble, trước hết ta cần tính thời gian thực hiện của swap. Dễ thấy thời gian thực hiện của swap là O(1) vì nó bao gồm chỉ 3 lệnh gán.

Trong Bubble, lệnh {3} gọi Swap nên chỉ tốn O(1), lệnh {2} thực hiện n-i lần, mỗi lần tốn O(1) nên chỉ tốn O(n-i). Lệnh {1} thực hiện n-1 lần nên T(n) = $\sum_{i=1}^{n-1}(n-i) = \frac{n(n-1)}{2} = O(n^2)$

5. Phân tích các chương trình đệ quy:

Với các chương trình có gọi các chương trình con đệ quy, ta không thể áp dụng cách tính như vừa trình bày ở mục 3.4 bởi vì một chương trình đệ quy sẽ gọi chính bản thân nó.

Với các chương trình đệ quy, trước hết ta cần thành lập các phương trình đệ quy, sau đó giải phương trình đệ quy, nghiệm của phương trình đệ quy sẽ là thời gian thực hiện của chương trình đệ quy.

a. Thành lập phương trình đệ quy:

Phương trình đệ quy là một phương trình biểu diễn mối liên hệ giữa T(n) và T(k), trong đó T(n) là thời gian thực hiện chương trình với kích thước dữ liệu nhập là n, T(k) thời gian thực hiện chương trình với kích thước dữ liệu là k, với k<n. Để thành lập được phương trình đệ quy, ta phải căn cứ vào chương trình đệ quy.

Ví dụ:

```
int giaithua (int n)
{
 if (n= =0)
        giaithua=1
 else
        giaithua= n*giaithua(n-1)
}
```

Gọi T(n) là thời gian thực hiện việc tính n giai thừa, thì T(n-1) là thời gian thực hiện việc tính n-1 giai thừa. Trong trường hợp n=0 thì chương trình chỉ thực hiện một lệnh gán giaithua = 1, nên tốn O(1), do đó ta có T(0) = C1. Trong trường hợp n>0 chương trình phải gọi đệ quy giaithua (n-1), việc gọi đệ quy này tốn T(n-1), sau khi có kết quả của việc gọi đệ quy, chương trình phải nhân kết quả đó với n và gán cho giaithua. Thời gian để thực hiện phép nhân và phép gán là một hằng C2. Vậy ta có:

$$T(n) = \begin{cases} C1 \text{ nếu } n=0 \\ T(n-1) + C2 \text{ nếu } n>0 \end{cases}$$

Đây là phương trình đệ quy để tính thời gian thực hiện của chương trình đệ quy giaithua.

b. Giải phương trình đệ quy:

Có 3 phương pháp giải phương trình đệ quy

1. Phương pháp truy hồi

2. Phương pháp đoán nghiệm

3. Lời giải tổng quát của một lớp các phương trình đệ quy

Phương pháp truy hồi:

Dùng đệ quy để thay thế bất kỳ T(m) với m<n vào phía phải của phương trình cho đến khi tất cả T(m) với m>1 được thay thế bởi biểu thức của các T(1). Vì T(1) luôn là hằng số nên chúng ta có công thức của T(n) chứa các số hạng chỉ liên quan đến n và các hằng số

Ví dụ:

Giải phương trình $T(n) = \begin{cases} C1 \text{ nếu } n = 1 \\ 2T(n/2) + C_2 n \text{ nếu } n>1 \end{cases}$

Ta có $T(n) = 2T(\frac{n}{2}) + C_2 n$

$T(n) = 2[2T(\frac{n}{4}) + C_2 \frac{n}{2}] + C_2 n = 4T(\frac{n}{4}) + 2C_2 n$

$T(n) = 4[2T(\frac{n}{8}) + C_2 \frac{n}{4}] + 2C_2 n = 8T(\frac{n}{8}) + 3C_2 n$

$T(n) = 2^i 8T(\frac{n}{2^i}) + iC_2 n$

Giả sử $n= 2^k$, quá trình suy rộng sẽ kết thúc khi i= k, khi đó ta có:

$T(n) = 2^k T(1) + kC_2 n$

Vì $2^k = n$ nên k = logn và với T(1) =C1 nên ta có:

$T(n) = C_1 n + C_2 n\log n$

Hay T(n) là O(nlogn)

Đoán nghiệm (Sinh viên có thể xem tham khảo)

Ta đoán một nghiệm f(n) và dùng chứng minh quy nạp để chứng tỏ rằng $T(n) \le f(n)$ với mọi n.

Thông thường f(n) là một trong các hàm quen thuộc như logn, n, nlogn, n^2, n^3, 2^n, n!, n^n

Đôi khi chúng ta chỉ đoán dạng của f(n) trong đó có một vài tham số chưa xác định (chẳng hạn $f(n) = an^2$ với a chưa xác định) và trong quá trình chứng minh quy nạp ta sẽ suy diễn ra giá trị thích hợp của các tham số.

Ví dụ:

Giải phương trình

$$T(n) = \begin{cases} C1 \text{ nếu } n = 1 \\ 2T(n/2) + C_2 n \text{ nếu } n>1 \end{cases}$$

Giả sử chúng ta đoán f(n) = anlogn. Với n = 1 ta thấy rằng cách đoán như vậy không được bởi vì anlogn có giá trị 0 không phụ thuộc vào giá trị của a. Vì thế ta thử tiếp theo f(n) = anlogn +b

Với n=1 ta có, $T(1) = C_1$ và f(1)=b, muốn $T(1) \le f(1)$ thì $b \ge C_1$ (*)

Giả sử rằng $T(k) \le aklogk + b$ với mọi k<n. Ta sẽ chứng minh $T(n) \le anlogn + b$

Giả sử $n \ge 2$, từ (I.1) ta có $T(n) = 2T(n/2)+C_2 n$

Áp dụng (I.2) với k= n/2 <n ta có:

$T(n) = 2T(n/2)+ C_2n \le 2[an/2\log(n/2)+b]+C_2n$

$T(n) \le an\log n - an + 2b + C_2n$

$T(n) \le (an\log n + b) + [b+(C_2-a)n]$. Nếu lấy $a \ge C_2 + b$ (**) ta được

$T(n) \le (an\log n + b) + [b+(C_2-C_2-b)n]$

$T(n) \le (an\log n + b) + (1-n)b$

$T(n) \le an\log n + b$

Nếu ta lấy a và b sao cho cả (*) và (**) đều thỏa mãn thì $T(n) \le an\log n + b$ với mọi nb

Ta phải giải hệ $\begin{cases} b \ge C_1 \\ a \ge C_2 + b \end{cases}$ để đơn giản, ta giải hệ $\begin{cases} b = C_1 \\ a = C_2 + b \end{cases}$

Dễ dàng ta có $b = C_1$ và $a = C_1 + C_2$ ta được $T(n) \le (C_1 + C_2)\, n\log n + C_1$ với mọi n

Hay nói cách khác T(n) là O(nlogn)

Lời giải tổng quát cho một lớp các phương trình đệ quy

Để giải bài tóan kích thước n, ta chia bài tóan đã cho thành a bài tóan con, mỗi bài tóan con có kích thước n/b. Giải các bài tóan con này và tổng hợp kết quả lại để được kết quả của bài tóan đã cho. Với các bài tóan con ta cũng làm như vậy. Kỹ thuật này sẽ dẫn chúng ta đến một chương trình đệ quy

Giả thuyết rằng mỗi bài toán con kích thước 1 lấy một đơn vị thời gian và thời gian để chia bài tóan kích thước n thành các bài tóan con kích thước n/b và tổng hợp kết quả từ các bài tóan con để được lời giải của bài tóan ban đầu là d(n).

Gọi T(n) là thời gian để giải bài tóan kích thước n thì ta có phương trình đệ quy

$$\begin{cases} T(1) = 1 \\ T(n) = aT(\frac{n}{b}) + d(n) \end{cases} \quad \text{(I.1)}$$

Ta sử dụng phương pháp truy hồi để giải phương trình này

$T(n) = aT(n/b) + d(n)$

$$T(n) = a[aT\left(\frac{n}{b^2}\right) + d\left(\frac{n}{b}\right)] + d(n) = a^2T\left(\frac{n}{b^2}\right) + ad\left(\frac{n}{b}\right) + d(n)$$

$$T(n) = a^2[aT\left(\frac{n}{b^3}\right) + d\left(\frac{n}{b^2}\right)] + ad\left(\frac{n}{b}\right) + d(n) = a^3T\left(\frac{n}{b^3}\right) + a^2d\left(\frac{n}{b^2}\right) + ad\left(\frac{n}{b}\right) + d(n)$$

$= \ldots$

$$= a^i T\left(\frac{n}{b^i}\right) + \sum_{j=0}^{i-1} a^j d\left(\frac{n}{b^j}\right)$$

Giả sử $n = b^k$ ta được: $T(n/b^k) = T(1) = 1$. Thay vào trên với i= k ta có:

$$T(n) = a^k + \sum_{j=0}^{k-1} a^j d\left(b^{k-j}\right) \quad \text{(I.2)}$$

Hàm tiến triển, nghiệm thuần nhất và nghiệm riêng

Trong phương trình đệ quy (I.1) hàm thời gian d(n) được gọi là hàm tiến triển (driving function)

Trong công thức (I.2), $a^k = a^{\log_b a}$ được gọi là nghiệm thuần nhất (homogeneous solutions)

Nghiệm thuần nhất là nghiệm chính xác khi d(n) = 0 với mọi n. Nói cách khác, nghiệm thuần nhất biểu diễn thời gian để giải tất cả các bài tóan con.

Trong công thức (I.2), $\sum_{j=0}^{k-1} a^j d\left(b^{k-j}\right)$ được gọi là nghiệm riêng (particular solutions)

Nghiệm riêng biểu diễn thời gian phải trả để tạo ra các bài tóan con và tổng hợp các kết quả của chúng. Nhìn vào công thức ta thấy nghiệm riêng phụ thuộc vào hàm tiến triển, số lượng và kích thước các bài tóan con.

Khi tìm nghiệm của phương trình (I.1), chúng ta phải tìm nghiệm riêng và so sánh với nghiệm thuần nhất. Nếu nghiệm riêng nào lớn hơn, ta lấy nghiệm đó làm nghiệm của phương trình (I.1)

Việc xác định nghiệm riêng không đơn giản chút nào, tuy vậy chúng ta cũng tìm được một lớp các hàm tiến triển có thể dễ dàng xác định nghiệm riêng.

Hàm nhân

Một hàm f(n) được gọi là hàm nhân (multiplicative function) nếu f(m.n)=f(m).f(n) với mọi số nguyên dương m và n.

Ví dụ:

Hàm $f(n) = n^k$ là một hàm nhân, vì $f(m.n) = (m.n)^k = m^k.n^k = f(m).f(n)$

Nếu d(n) trong (I.1) là một hàm nhân thì theo tính chất của hàm nhân ta có $d(b^{k-j}) = (d(b))^{k-j}$ và nghiệm riêng của (I.2) là:

$$\sum_{j=0}^{k-1} a^j d(b^{k-j}) = d(b)^k \sum_{j=0}^{k-1} \left[\frac{a}{d(b)}\right]^j = d(b)^k \frac{\left[\frac{a}{d(b)}\right]^k - 1}{\frac{a}{d(b)} - 1} = \frac{a^k - d(b)^k}{\frac{a}{d(b)} - 1} \quad \text{(I.3)}$$

Xét 3 trường hợp sau:

1. Nếu a> d(b) thì nghiệm riêng là $O(a^k) = O(n^{\log_b a})$. Như vậy nghiệm riêng và nghiệm thuần nhất bằng nhau do đó $T(n) = O(n^{\log_b a})$

Ta cũng thấy thời gian thực hiện chỉ phụ thuộc vào a, b mà không phụ thuộc vào hàm tiến triển d(n). Vì vậy để cải tiến giải thuật ta cần giảm a hoặc tăng b

2. Nếu a <d(b) thì nghiệm riêng là $O(d(b)^k) = O(n^{\log_b d(b)})$. Trong trường hợp này nghiệm riêng lớn hơn nghiệm thuần nhất nên $T(n) = O(n^{\log_b d(b)})$

Để cải tiến giải thuật chúng ta cần để ý đến cả d(n), a và b cùng mức độ như nhau

Trường hợp đặc biệt quan trọng khi $d(n) = n^\alpha$. Khi đó $d(b) = b^\alpha$ và $\log_{b(}b^\alpha) = \alpha$. Vì thế nghiệm riêng là $O(n^\alpha)$ và do vậy T(n) là $O(n^\alpha)$

3. Nếu a = d(b) thì công thức (I.5) không xác định nên ta tính trực tiếp nghiệm riêng:

Nghiệm riêng =

$$\sum_{j=0}^{k-1} a^j (d(b))^{k-j} = d(b)^k \sum_{j=0}^{k-1} \left[\frac{a}{d(b)}\right]^j = d(b)^k \sum_{j=0}^{k-1} 1 = d(b)^k k = n^{\log_b d(b)} \log_b n$$

Vì a = d(b) nên nghiệm riêng là $n^{\log_b d(b)} \log_b n$ và nghiệm này lớn gấp $\log_b n$ lần nghiệm thuần nhất. Do đó $T(n)= O(n^{\log_b a} \log_b n)$

Trong trường hợp đặc biệt $d(n) =n^\alpha$ ta được $T(n) =O(n^\alpha \log n)$

Chú ý khi giải một phương trình đệ quy cụ thể, ta phải xem phương trình đó có thuộc dạng phương trình tổng quát hay không. Nếu có thì phải xem xét hàm tiến triển có phải là hàm nhân không. Nếu có thì ta xác định a, d(b) và dựa vào sự so sánh giữa a và d(b) mà vận dụng một trong ba trường hợp nói trên.

Ví dụ:

Giải các phương trình đệ quy sau với T(1) =1 và

1. $T(n) = 4T(n/2) + n$

2. $T(n) = 4T(n/2) + n^2$

3. $T(n) = 4T(n/2) +n^3$