BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT HƯNG YÊN

ĐỀ CƯƠNG BÀI GIẢNG

PHÂN TÍCH VÀ THIẾT KẾ THUẬT TOÁN

TRÌNH ĐỘ ĐÀO TẠO: **ĐẠI HỌC CHÍNH QUY**

NGÀNH ĐÀO TẠO: **CÔNG NGHỆ THÔNG TIN**

(INFORMATION TECHNOLOGY)

Hưng Yên, năm 2021

MỤC LỤC

LỜI NÓI ĐẦU

Những kiến thức về thuật toán và cách thiết kế, đánh giá thuật toán đóng vai trò quan trọng trong việc đào tạo cử nhân, kỹ sư công nghệ thông tin. Ngoài việc học phân tích và thiết kế thuật toán, người học còn được cung cấp những kiến thức, kỹ năng cần thiết giải các bài toán hay gặp trong tin học để trở thành người lập trình viên chuyên nghiệp.

Về nội dung, cuốn sách này chia thành các bài tương ứng sát với chương trình học của sinh viên khoa Công nghệ thông tin. Sách trình bày những chiến lược thiết kế thuật toán quan trọng như: tham lam, chia-để-trị, quy hoạch động, nhánh cận, quay lui, và những thuật toán dựa trên kinh nghiệm. Trong mỗi chiến lược thiết kế, bên cạnh việc đào sâu phân tích, chúng còn được thảo luận về độ phức tạp thông qua các bài toán cụ thể. Ngoài ra, còn có các bài tập thực hành và hệ thống các bài kiểm tra cài đặt các thuật toán trên. Trong tài liệu này sử dụng ngôn ngữ C# để minh họa, cài đặt. Tuy nhiên người học dễ dàng cài đặt được bằng các ngôn ngữ lập trình khác như: VB.NET, C/C++, Pascal... do có mô tả thuật toán bằng mã giả.

Khoa Công nghệ thông tin

BÀI 1: THUẬT TOÁN VÀ CÁC VẤN ĐỀ LIÊN QUAN

1.1. Giới thiệu môn học, phương pháp học

Đây là module cung cấp cho người học những kỹ thuật và công cụ cơ bản cho việc phân tích và thiết kế các thuật thuât. Modul trình bày cách thiết kế và phân tích những bài toán sang mô hình toán học để rồi đi tới các lời giải theo thuật ngữ lập trình.

Modul cũng trình bày những chiến lược thiết kế thuật toán quan trọng như: tham lam, chia-để-trị, quy hoạch động, nhánh cận, quay lui, và những thuật toán dựa trên kinh nghiệm. Trong mỗi chiến lược thiết kế, bên cạnh việc đào sâu phân tích, chúng còn được thảo luận về độ phức tạp thông qua các bài toán cụ thể.

Ngoài việc học phân tích và thiết kế thuật toán, người học còn được cung cấp những kiến thức, kỹ năng cần thiết giải các bài toán hay gặp trong tin học để trở thành người lập trình viên chuyên nghiệp.

Module này sử dụng ngôn ngữ C# để minh họa, cài đặt. Tuy nhiên người học dễ dàng cài đặt được bằng các ngôn ngữ lập trình khác như: VB.NET, C/C++, Pascal...

Sau khi hoàn thành module này, người học có khả năng:

- Trình bày được các khái niệm và kỹ thuật để thiết kế một thuật toán hiệu quả.
- Thiết lập được các phương trình đệ qui để mô tả độ phức tạp về thời gian và không gian của một thuật toán đệ qui.
- Xác định được độ phức tạp về thời gian và không gian của thuật toán xác định, trong các trường hợp: tốt nhất, trung bình cũng như xấu nhất.
- Nhận ra các bài toán mà ở đó dùng phương pháp quy hoạch động là một giải pháp hiệu quả.
- Áp dụng kỹ thuật chia-để-trị vào việc thiết kế bài toán cũng như phân tích độ phức tạp tính toán.

- Cài đặt các thuật toán tham lam (gready), chia để trị (divide-and-conquer), quay lui (backtracking), nhánh-cận (branch-and-bound), và quy hoạch động (dynamic programming) cho một số bài toán.
- Chuyển đổi hiệu quả được từ sơ đồ thuật toán sang chương trình.
- Áp dụng những kỹ thuật, chiến lược thiết kế thuật toán cho bài toán: Lập kế hoạch cho một quy trình sản xuất, tìm đường đi trên bản đồ phức tạp, viết chương trình Game,...
- Áp dụng những kỹ thuật thiết kế thuật toán nhằm tăng hiệu năng cho những dự án cụ thể trong khi phát triển phần mềm.
- Làm việc nhóm trong quá trình xây dựng bài tập lớn để đạt được các mục tiêu trên.

Để học tốt môn học này mỗi người học phải tự xây dựng cho mình một phương pháp học thích hợp. Nhưng phương pháp chung để học môn học này là người học phải hiểu thật kỹ các phần lý thuyết cơ sở, rèn luyện sự tư duy logic một vấn đề và vận dụng một cách linh hoạt vào các trường hợp cụ thể, cài đặt các thuật toán….

1.2. Khái niệm thuật toán

1.2.1. Khái niệm thuật toán:

Thuật toán (algorithm) là một trong những khái niệm quan trọng trong lĩnh vực tin học. Thuật ngữ thuật toán được xuất phát từ nhà toán học Arập Abu Ja'far Mohammed ibn Musa al Khowarizmi (khoảng năm 825). Tuy nhiên lúc bấy giờ và trong nhiều thế kỷ sau, nó không mang nội dung như ngày nay chúng ta quan niệm. Thuật toán nổi tiếng nhất có từ thời cổ Hy lạp là thuật toán Euclid, thuật toán tìm ước chung lớn nhất của hai số nguyên. Có thể mô tả thuật toán đó như sau:

Thuật toán Euclid.

Input: **m, n** nguyên dương

Output: **g** (ước chung lớn nhất của m và n)

Phương pháp:

Bước 1: Tìm **r**, phần dư của **m** cho **n**

Bước 2: Nếu **r** = 0, thì **g:=n** (gán giá trị của n cho g),và dừng lại.

Trong trường hợp ngược lại (r≠0), thì **m:=n; n:=r** và quay lại bước 1.

Chúng ta có thể quan niệm các bước cần thực hiện để làm một món ăn, được mô tả trong các sách dạy chế biến món ăn, là một thuật toán. Cũng có thể xem các bước cần tiến hành để gấp đồ chơi bằng giấy, được trình bày trong sách dạy gấp đồ chơi bằng giấy là một thuật toán. Phương pháp cộng nhân các số nguuyên, chúng ta đã được học ở cấp I cũng là các thuật toán.

Vì vậy ta có định nghĩa không hình thức về thuật toán như sau:

Thuật toán là một dãy hữu hạn các bước, mỗi bước mô tả chính xác các phép toán, hoặc hành động cần thực hiện. .. để cho ta lời giải của bài toán.

(Từ điểm Oxford Dictionary định nghĩa, Algorithm: set of well - defined rules for solving a problem in a finite number of steps.)

1.2.2. Các yêu cầu về thuật toán

Để hiểu đầy đủ ý nghĩa của khái niệm thuật toán, chúng ta đưa ra 5 đặc trưng sau đây của thuật toán.

1. Input

Mỗi thuật toán đều có một số (có thể bằng không) các dữ liệu vào (input). Đó là các giá trị cần đưa vào khi thuật toán bắt đầu làm việc. Các dữ liệu này cần được lấy từ các tập hợp giá trị cụ thể nào đó. Chẳng hạn, trong thuật toán Euclid ở trên, các số m và n là các dữ liệu lấy từ tập các số nguyên dương.

2. Output

Mỗi thuật toán cần có một hoặc nhiều dữ liệu ra (output). Đó là các giá trị có quan hệ hoàn toàn xác định với các dữ liệu vào, và là kết quả của sự thực hiện thuật toán. Trong thuật toán Euclid, có một dữ liệu ra đó là ƯSCLN g, khi thuật toán dừng lại (trường hợp r=0) thì giá trị của g là ước chung lớn nhất của m và n.

3. Tính xác định.

Ở mỗi bước, các bước thao tác phải hết sức rõ ràng, không gây nên sự nhập nhằng. Nói rõ hơn là trong cùng một điều kiện hai bộ xử lý cùng thực hiện một thuật toán phải cho cùng một kết quả như nhau. Nếu biểu diễn thuật toán bằng phương pháp thông thường không có gì đảm bảo được ngươì đọc hiểu đúng ý của người viết thuật toán. Để đảm bảo đòi hỏi này, thuật toán cần được mô tả trong các ngôn ngữ lập trình (ngôn ngữ máy, hợp ngữ hoặc ngôn ngữ bậc cao như Pascal..). Trong các ngôn ngữ này các mệnh đề được tạo theo các qui tắc cú pháp nghiêm ngặt và chỉ có một nghĩa duy nhất.

4. Tính khả thi.

Tất cả các phép toán có mặt trong thuật toán phải đủ đơn giản. Điều đó có nghĩa là, các phép toán phải sao cho, ít nhất về nguyên tắc có thể thực hiện bởi con người chỉ bằng giấy trắng và bút chì trong một khoảng thời gian hữu hạn. Chẳng hạn, trong thuật toán Euclid ta chỉ cần thực hiện các phép chia các số nguyên, các phép gán và các phép so sánh r=0 hay $r \neq 0$. Điều quan trọng nữa là thuật toán phải có tính đa năng làm việc được với tất cả các tập hợp dữ liệu có thể của đầu vào.

5. Tính dừng.

Với mọi bộ dữ liệu vào thoả mãn các điều kiện của dữ liệu vào (tức là được lấy ra từ các tập của dữ liệu vào), thuật toán phải dừng lại sau một số hữu hạn bước thực hiện.Ví dụ, thuật toán Euclid thoả mãn điêù kiện này. Bởi vì giá trị của r luôn nhỏ hơn n (khi thực hiện bước 1), nếu r <>0 thì giá trị của n ở bước i (ký hiệu là n_i) sẽ là gía trị của r_{i-1} ở bước i-1, ta phải có bất đẳng thức $n>r = n_1 > r_1 = n_2 > r_2$. Dãy số nguyên dương này giảm dần và cần phải kết thúc ở 0, do đó sau một số hữu hạn bước nào đó giá trị của r phải = 0 và thuật toán phải dừng lại.

Với một vấn đề đặt ra, có thể có một hoặc nhiều thuật toán giải. Một vấn đề có thuật toán giải gọi là vấn đề giải được (bằng thuật toán). Chẳng hạn, tìm nghiệm của hệ phương trình tuyến tính là vấn đề giải được. Một vấn đề không tồn tại thuật toán gọi là vấn đề không giải được (bằng thuật toán). Một trong những thành tựu suất xác nhất của toán học thế kỷ 20 là đã tìm ra những vấn đề không giải được bằng thuật toán. Chẳng hạn thuật toán chắc thắng cho người thứ hai của cờ ca rô hoặc thuật toán xác định xem một máy Turing có dừng lại sau n bước không, đều là những vấn đề không tồn tại thuật toán giải được.

1.3. Tại sao phải phân tích thuật toán

Giả sử, với một số bài toán nào đó chúng ta có một số thuật toán giải. Một câu hỏi mới xuất hiện là, chúng ta cần chọn thuật toán nào trong số các thuật toán đó để áp dụng. Việc phân tích thuật toán, đánh giá độ phức tạp của thuật toán là nội dung của phần dưới đây sẽ giải quyết vấn đề này.

1.4. Những vấn đề phát sinh trong phân tích và thiết kế thuật toán

1.4.1. Thiết kế thuật toán

Để giải một bài toán trên máy tính điện tử (MTĐT), điều trước tiên là chúng ta phải có thuật toán. Một câu hỏi đặt ra là làm thế nào để tìm ra được thuật toán cho một bài toán đã đặt ra? Lớp các bài toán được đặt ra từ các ngành khoa học kỹ thuật, từ các lĩnh vực hoạt động của con người là hết sức phong phú và đa dạng. Các thuật toán giải các lớp bài toán khác nhau cũng rất khác nhau. Tuy nhiên, có một số kỹ thuật thiết kế thuật toán chung như: Chia để trị (divide-and-conque), phương pháp tham ăn (greedy method), qui hoạch động (dynamic programming)... Việc nắm được các chiến lược thiết kế thuật toán này là hết sức quan trọng và cần thiết, nó giúp cho ta dễ tìm ra các thuật toán mới cho các bài toán mới được đưa ra.

1.4.2. Tính đúng đắn của thuật toán.

Khi một thuật toán được làm ra, ta cần phải chứng minh rằng, thuật toán khi được thực hiện sẽ cho ta kết quả đúng với mọi dữ liệu vào hợp lệ. Điều này gọi là chứng minh tính đúng đắn của thuật toán. Việc chứng minh tính đúng đắn của thuật toán là một công việc không dễ dàng. Trong nhiều trường hợp, nó đòi hỏi ta phải có trình độ và khả năng tư duy toán học tốt.

Sau đây ta sẽ chỉ ra rằng, khi thực hiện thuật toán Euclid, g sẽ là ước chung lớn nhất của hai số nguyên dương bất kỳ m, n. Thật vậy, khi thực hiện bước 1, ta có m =

qn + r, trong đó q là số nguyên nào đó. Nếu r = 0 thì n là ước của m và hiển nhiên n (do đó g) là ước chung lớn nhất của m và n. Nếu $r \neq 0$ thì một ước chung bất kỳ của m và n cũng là ước chung của n và r (vì r=m-qn). Ngược lại một ước chung bất kỳ của n và r cũng là ước chung của m và n (vì m = qn + r). Do đó ước chung lớn nhất của n và r cũng là ước chung lớn nhất của ma và n. Vì vậy, khi thực hiện lặp lại bước 1, với sự thay đổi giá trị của m bởi n, và sự thay đổi giá trị của n bởi r, cho tới khi r=0 ta nhận được giá trị của g là ước chung lớn nhất của các giá trị m và n ban đầu.

1.4.3. Phân tích thuật toán

Giả sử, với một số bài toán nào đó chúng ta có một số thuật toán giải. Một câu hỏi mới xuất hiện là, chúng ta cần chọn thuật toán nào trong số các thuật toán đó để áp dụng. Việc phân tích thuật toán, đánh giá độ phức tạp của thuật toán là nội dung của phần dưới đây sẽ giải quyết vấn đề này.

1.4.4. Đánh giá hiệu quả của thuật toán.

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

1. Thuật toán đơn giản, dễ hiểu, dễ cài đặt (dễ viết chương trình)
2. Thuật toán sử dụng tiết kiệm nhất các nguồn tài nguyên của máy tính, và đặc biệt chạy nhanh nhất có thể được.

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

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

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

Thời gian cần thiết để thực hiện thuật toán (ta gọi là thời gian chạy). Chúng ta chỉ quan tâm đến thời gian thực hiện thuậ toán, có nghĩa là ta nói đến đánh giá thời gian thực hiện. Một thuật toán có hiệu quả được xem là thuật toán có thời gian chạy ít hơn so với các thuật toán khác.

BÀI 2: PHÂN TÍCH VÀ THIẾT KẾ THUẬT TOÁN

2.1. Lý thuyết chung về phân tích thuật toán

2.1.1 Đặt vấn đề

Khi xây dựng được thuật toán để giải bài toán thì có hằng loạt vấn đề được đặt ra để phân tích. Thường là các vấn đề sau:

- Yêu cầu về tính đúng đắn của thuật toán, thuật toán có cho lời giải đúng

- Tính đơn giản của thuật toán. Thường ta mong muốn có được một thuật toán đơn giản, dễ hiểu, dễ lập trình. Đặc biệt là những thuật toán chỉ dùng một vài lần ta cần coi trọng tính chất này, vì công sức và thời gian bỏ ra để xây dựng thuật toán thường lớn hơn rất nhiều so với thời gian thực hiện nó.

- Yêu cầu về không gian: thuật toán được xây dựng có phù hợp với bộ nhớ của máy

- Yêu cầu về thời gian: Thời gian chạy của thuật toán có nhanh không? Một bài toán thường có nhiều thuật toán để giải, cho nên yêu cầu một thuật toán dẫn nhanh đến kết quả là một đòi hỏi đương nhiên… Trong phần này ta quan tâm chủ yếu đến tốc độ của thuật toán. Ta cũng lưu ý rằng thời gian chạy của thuật toán và dung lượng bộ nhớ nhiều khi không cân đối được để có một giải pháp trọn vẹn. Chẳng hạn, thuật toán sắp xếp trong sẽ có thời gian chạy nhanh hơn vì dữ liệu được lưu trữ trong bộ nhớ trong, và do đó không phù hợp trong trường hợp kích thước dữ liệu lớn. Ngược lại, các thuật toán sắp xếp ngoài phù hợp với kích thước dữ liệu lớn vì dữ liệu được lưu trữ chính ở các thiết bị ngoài, nhưng khi đó tốc độ lại chậm hơn.

2.1.2. Phân tích đánh giá thời gian chạy của thuật toán

- Bước đầu tiên phân tích thời gian chạy của thuật toán là quan tam đến kích thước dữ liệu như dữ liệu nhập của thuật toán và quyết định phân tích nào là thích hợp. Ta có thể xem thời gian chạy của thuật toán là một hàm theo kích thước của dữ liệu nhập. Nếu gọi n là kích thước của dữ liệu nhập thì thời gian thực hiện T của thuật toán được biểu diễn như một hàm theo n, ký hiệu là: T(n)

- Bước thứ hai trong việc phân tích đánh giá thời gian chạy của một thuật toán là nhận ra các thao tác trừu tượng của thuật toán để tách biệt sự phân tích và sự cài đặt. Bởi vì ta biết rằng tốc độ xử lý của máy tính và các bộ dịch của các ngôn ngữ lập trình cấp cao đều ảnh hưởng đến thời gian chạy của thuật toán, nhưng những yếu tố này ảnh hưởng không đồng đều với các loại máy trên đó cài đặt thuật toán, vì vậy không thể dựa vào chúng để đánh giá thời gian chạy của thuật toán. Ta thấy rằng T(n) không thể được biểu diễn bằng giây, phút...được; Cách tốt hơn là biểu diễn theo số chỉ thị của thuật toán

- Bước thứ ba trong việc phân tích đánh giá thời gian chạy của một thuật toán là phân tích về mặt toán học với mục đích tìm ra các giá trị trung bình và trường hợp xấu nhất cho mỗi đại lượng cơ bản. Chẳng hạn, khi sắp xếp một dãy các phần tử, thời gian chạy của thuật toán hiển nhiên còn phụ thuộc vào tính chất của dữ liệu nhập như:

Dãy có thứ tự đã sắp xếp

Dãy có thức tự ngược với thứ tự cần sắp xếp

Đã có thứ tự ngẫu nhiên

2.2. Sự tăng trưởng các hàm

2.2.1. Ký hiệu tiệm cận

Là ký hiệu dung để mô tả thời gian thực hiện tiệm cận của một thuật toán, nằm trong các trường hợp xấu nhất, trung bình, hoặc tốt nhất.
Đánh giá thời gian hay còn gọi là độ phức tạp của thuật toán được định nghĩa dưới dạng các hàm, trong đó miền xác định của các hàm đó là tập các số tự nhiên dựa trên kích thước dữ liệu đầu vào.

2.2.1. Ký hiệu tiệm cận Θ

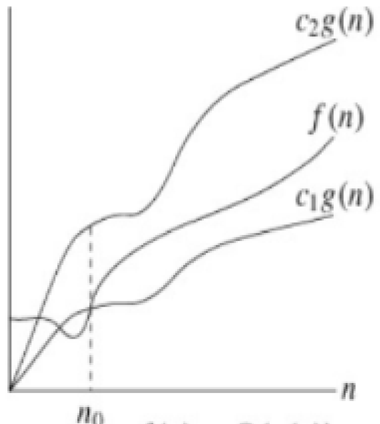
Với hàm g(n), Θ(g(n)) là tập hợp các hàm

$$\Theta(g(n))=\left\{ f(n) \exists c_1, c_2, n_0, 0'' c_1 g(n)'' f(n)'' c_2 g(n), \forall n \ge n_0 \right\}$$

Bởi vì Θ $(g(n))$ là một tập hợp nên ta có thể viết:

$f(n) \in \Theta(g(n))$ Hoặc $f(n) = \Theta(g(n))$

Với $n >= n_0$, $c_1\, g(n) \le f(n) \le c_2\, g(n)$. Ta nói rằng g(n) là tiệm cận của f(n)



$f(n) = \Theta(g(n))$

2.2.2. Ký hiệu tiệm cận trên O

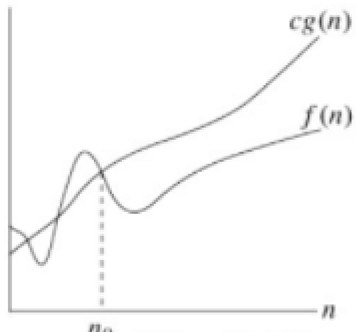
Các ký hiệu Θ là ký hiệu tiệm cận giới hạn của một hàm chặn trên và chặn dưới. Khi chỉ có một tiệm cận chặn trên thì ta sử dụng ký hiệu O.

Định nghĩa: Với một hàm đã cho g(n), ta định nghĩa:

$O(g(n)) = \{f(n)$: tồn tại hằng số dương c và n_0 sao cho:

$$0 \le f(n) \le cg(n), \text{ với mọi } n \ge n_0\}$$

Với mọi giá trị $n \ge n_0$, giá trị của hàm $f(n) \le cg(n)$.



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

+ Khi viết $f(n) = O(g(n))$ nghĩa là $f(n)$ là một phần tử của tập $O(g(n))$.

+ Lưu ý rằng $f(n) = \Theta(g(n))$ suy ra $f(n) = O(g(n))$, bởi vì ký hiệu Θ là một khái niệm mạnh hơn ký hiệu O.

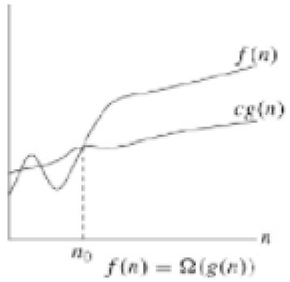
2.2.3. Tiệm cận dưới Ω

Định nghĩa: $\Omega(g(n)) = \{f(n)$: tồn tại hằng số dương c và n_0 sao cho

$0 \le c.g(n) \le f(n)$, với mọi $n \ge n_0$).

Định lý: Với hai hàm bất kỳ $f(n)$ và $g(n)$, $f(n)=\Theta(g(n))$ khi và chỉ khi $f(n) = O(g(n))$ và $f(n) = \Omega(g(n))$.

Do hệ ký hiệu Ω mô tả một giới hạn dưới, ta cũng có thể sử dụng nó để giới hạn thời gian thực hiện trong trường hợp tốt nhất của một thuật toán.



2.3. Các quy tắc đánh giá độ phức tạp thuật toán

2.3.1. O(f(x)) và đánh giá thời gian thực hiện thuật toán.

Khi đánh giá thời gian thực hiện bằng phương pháp toán học, chúng ta sẽ bỏ qua nhân tố phụ thuộc vào cách cài đặt chỉ tập trung vào xác định độ lớn của thời gian thực hiện T(n).

Giả sử n là số nguyên không âm. T(n) và f(n) là các hàm thực không âm. Ta viết T(n)=O(f(n)) (đọc T(n) là ô lớn của f(n)), nếu và chỉ nếu tồn tại các hằng số dương c và no sao cho $T(n) \le c.f(n)$, với mọi $n \ge no$.

Nếu một thuật toán có thời gian thực hiện T(n) = O(f(n)), ta nói thuật toán có thời gian thực hiện cấp f(n). Từ định nghĩa ký hiệu ô lớn, ta có thể xem rằng hàm f(n) là cận trên của T(n).

Ví dụ. Giả sử $T(n) = 3n^2 + 4n +5$. Ta có

$3n^2 + 4n + 5 \le 3n^2 + 4n^2 + 5n^2 = 12n^2$, với mọi $n \ge 1$.

Vậy $T(n) = O(n^2)$. Trong trường hợp này ta nói thuật toán có thời gian thực hiện cấp n^2, hoặc gọn hơn, thuật toán có thời gian thực hiện bình phương.

Dễ dàng thấy được, nếu T(n)= O(f(n)) và $f(n)= O(f_1(n))$, thì $T(n) = O(f_1(n))$. Thật vậy, vì T(n) là ô lớn của f(n) và f(n) là ô lớn của f1(n) nên tồn tại các hằng số c_o, n_o,c_1, n_1 sao cho $T(n) \le c_0 f(n)$ với mọi $n \ge n_0$ và $f(n) \le c_1 f_1(n)$ với mọi $n \ge n_1$. Từ đó ta có $T(n) \le c_0c_1f_1(n)$ với mọi $n \ge \max(n_0, n_1)$.

Khi biểu diễn cấp của thời gían thực hiện thuật toán bởi hàm f(n), chúng ta sẽ chọn f(n) là hàm nhỏ nhất, đơn giản nhất có thể được sao cho T(n) = 0(f(n)). Thông thường f(n) là các hàm số sau đây: f(n)=1 ; f(n)= logn; f(n) =n; f(n) = nlog(n) ; f(n)= n^2; n^3 … ; $f(n) = 2^n$.

- Nếu T(n)= O(1) điều này có nghĩa là thời gian thực hiện thuật toán được chặn trên bởi một hằng nào đó, trong trường hợp này ta nói thuật toán có thời gian *thực hiện hằng*.

- Nếu T(n)= O(n), tức là bắt đầu từ một n_0 nào đó trở đi ta có T(n) ≤ cn với một hằng số c nào đó, trong trường hợp này ta nói thuật toán có thời gian *thực hiện tuyến tính.*

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

Ký hiệu O(f(x)) của f(x)	Độ phức tạp loại
$O(1)$	Hằng
O(log)	Logarit
$O(n)$	Tuyến tính
O(nlogn)	n log n
$O(n^2)$	Bình ph- ơng
$O(n^3)$	Lập ph- ơng
O(2^n)	Mũ
$O(n!)$	Giai thừa

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

- Các hàm loại : 2^n, n!, nn thường được gọi là các hàm loại mũ. Thuật toán với thời gian chạy có cấp hàm loại mũ thì tốc độ rất chậm

- Các hàm n, n^3, n^2, nlog $_2$ n thường được gọi là các hàm đa thức. Thuật toán với thời gian chạy có cấp hàm đa thức thường chấp nhận được

2.3.2. Các qui tắc để đánh giá thời gian thực hiện thuật toán

Sau đây là qui tắc cần thiết về ô lớn để đánh giá thời gian thực hiện thuật toán.

Qui 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 $T_1(n)$, $T_2(n)$ lần lượt là ô lớn của $f_1(n)$ và $f_2(n)$ tương ứng do đó tồn tại hằng số c_1, c_2, n_1, n_2 sao cho $T_1 (n) \le c_1f_1(n)$, $T_2(n) \le c_2f_2(n)$ với mọi $n \ge n_1$, mọi $n \ge n_2$. Đặt $n_0 = \max (n_1,n_2)$.

Khi đó, với mọi $n \ge n0$ ta có bất đẳng thức sau:

$$T_1(n) + T_2(n) \le (c_1 + c_2) \max (f_1(n), f_2(n)).$$

Qui tắc này thường được áp dụng như sau. Giả sử thuật toán của ta được phân thành ba phần tuần tự. Phần một có thời gian thực hiện $T_1(n)$ được đánh giá là O(1), phần hai có thời gian thực hiện là $T_2(n)$ và có thời gian đánh gía là $O(n^2)$, phần ba có thời gian thực hiện $T_3(n)$ có thời gian đánh giá là O(n). Khi đó thời gian thực hiện thuật toán là $T(n) = T_1(n) + T_2(n) + T_3(n)$ là $O(n_2)$, vì $n_2 = max(1, n^2, n)$.

Trong sách báo quốc tế các sách báo thường được trình bày dưới dạng các thủ tục hoặc hàm trong ngôn ngữ tựa Pascal. Để đánh giá thời gian thực hiện thuật toán ta cần biết cách đánh giá thời gian thực hiện các câu lệnh trong Pascal, các câu lệnh trong Pascal được định nghĩa đệ qui như sau:

1. Các phép gán, đọc, viết, goto là câu lệnh. Các lệnh này được gọi là các lệnh đơn.

Thời gian thực hiện các lệnh đơn là O(1).

2. Nếu S1, S2, ..., Sn là câu lệnh thì

begin S1, S2,. ..., Sn **end**

là câu lệnh.

Lệnh này được gọi là lệnh hợp thành (hoặc khối).

Thời gian thực hiện lệnh hợp thành được xác định bởi luật tổng.

3. Nếu S1, S2 là các câu lệnh và E là biểu thức logic thì :

If E **then** S1 **else** S2

Và

if E **then** S1

là câu lệnh. Các lệnh này được gọi là lệnh **if.**

Đánh giá thời gian thực hiện các lệnh **if** : Giả sử thời gian thực hiện các lệnh S1,S2, là $O(f_1(n))$ và $O(f_2(n))$ tương ứng. Khi đó thời gian thực hiện lệnh if là : $O(max(f_1(n),f_2(n)))$.

4. Nếu S1,S2,. ..., Sn là các câu lệnh, E là biểu thức có kiểu thứ tự đếm được, và v1,v2,. ..., vn là các giá trị có cùng kiểu với E thì :

Case E of

v1: S1 ;

v2: S2 ;

....

vn : Sn;;

end;

là các lệnh.

Lệnh này được gọi là lệnh **case.**

Đánh giá thời gian thực hiện lệnh case như lệnh if

5. Nếu S là các câu lệnh và E là biểu thức logíc thì

While E do S

Là câu lệnh. Lệnh này được gọi là lệnh while.

Thời gian thực hiện lệnh while được đánh giá : Giả sử thời gian thực hiện lệnh S (thân của lệnh while) là O(f(n)). Giả sử g(n) là số tối đa các lần thực hiện lệnh S, khi thực hiện lệnh while. Khi đó thời gian thực hiện lệnh while là O(f(n)g(n)).

Nếu S1, S2,....., Sn là các câu lệnh, E là biểu thức logíc thì

Repeat S1, S2,. ., Sn until E

Là câu lệnh. Lệnh này được gọi là lệnh repeat.

Giả sử, thời gian thực hiện khối begin S1, S2,…Sn end; là O(f(n)). Giả sử g(n) là số tối đa các lần lặp. Khi đó thời gian thực hiện lệnh repeat là O(f(n),g(n)).

Với S là câu lệnh và E1,E2 là biểu thức cùng một kiểu thứ tự đếm được thì

For i:= E1 to E2 do S   là câu lệnh, và

for i:= E2 downto E1 do S là câu lệnh.

Các câu lệnh này được gọi là lệnh for.

Thời gian thực hiện lệnh for được đánh giá tương tự như thời gian thực hiện lệnh while và lệnh repeat.

BÀI 3. THUẬT TOÁN CHIA ĐỂ TRỊ

3.1. Đệ quy

Giải thuật đệ qui là giải thuật mà trong quá trình mô tả giải thuật ta lại sử dụng chính nó với qui mô thu hẹp hơn.

Ví dụ 1: n! = n(n - 1)!

Ví dụ 2: Tính số n trong dãy Fibonacy

F0 = F1=1

Fn = Fn - 1+ Fn - 2 với n ≥ 2.

Rõ ràng, đệ quy mạnh ở chỗ có thể định nghĩa một tập vô hạn các đối tượng chỉ bởi một số hữu hạn các mệnh đề. Cũng bằng cách đó, một số vô hạn các phép tính có thể được mô tả bởi một chương trình đệ quy, mặc dù trong chương trình không có các vòng lặp tường minh. Tuy nhiên thuật giải đệ quy thích hợp khi các bài toán, các hàm hay các cấu trúc dữ liệu cũng được định nghĩa theo kiểu đệ quy.

3.2. Định lý Master

Chúng ta sử dụng Định lý thợ (Master Theorem) để giải các công thức đệ quy dạng sau một cách hiệu quả :

T(n) =aT(n/b) + c.n^k trong đó a>=1, b>1

• Bài toán ban đầu được chia thành a bài toán con có kích thước mỗi bài là n/b, chi phí để tổng hợp các bài toán con là f(n).

• Ví dụ : Thuật toán sắp xếp trộn chia thành 2 bài toán con, kích thước n/2. Chi phí tổng hợp 2 bài toán con là O(n).

Định lý thợ a>=1, b>1, c, k là các hằng số. T(n) định nghĩa đệ quy trên các tham số không âm

T(n) = aT(n/b) + c.n^k

+ Nếu a> b^k thì T(n) =O(n^ (logab))

+ Nếu a= b^k thì T(n)=O(n^k.lgn)

+ Nếu a< b^k thì T(n) = O(n^k)

Chú ý: Không phải trường hợp nào cũng áp dụng được định lý thợ.

Ví dụ: T(n) = 2T(n/2) +nlogn a =2, b =2, nhưng không xác định được số nguyên k

Nhận xét: Có thể hiểu là ta chia bài toán lớn ra thành a bài toán có kích thước n/b để giải rồi dùng f(n) phép tính để kết hợp lời giải các bài toán còn lại.

Ví dụ:

a. Thuật toán nhân số nguyên n-bit của Karatsuba-Ofman có độ phức tạp thuật toán

$$T(n) = 3T(n/2) + O(n) = \Theta(n^{\log_2 3}) = \Theta(n^{1,58})$$

theo trường hợp 1.

b. Nếu

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

thì theo trường hợp 2 có độ phức tạp thuật toán

$$T(n) = \Theta(n^2)$$

c. Thuật toán sắp xếp trộn (Merge Sort) có độ phức tạp thuật toán

$$T(n) = 2T(n/2) + O(n) = \Theta(n \log_2 n)$$

thì theo trường hợp 3.

3.3. Cơ bản về thuật toán chia để trị

Chia để trị là một kỹ thuật thiết kế thuật toán bao gồm việc chia một bài toán cần giải ra thành những bài toán con nhỏ hơn có cùng một loại vấn đề, giải từng bài toán con đó một cách lần lượt và độc lập, sau đó kết hợp các lời giải con thu được nhờ cách đó để thu được lời giải của bài toán nguyên thuỷ. Hai câu hỏi tự nhiên nảy ra là "Vì sao ai đó làm việc này?" và "Chúng ta cần giải các bài toán con như thế nào?". Tính hiệu quả của kỹ thuật chia để trị nằm ở câu trả lời cho câu hỏi thứ hai.

3.3.1 Ý tưởng thuật toán:

Có lẽ quan trọng và áp dụng rộng rãi nhất là kỹ thuật thiết kế “Chia để trị”. Nó phân rã bài toán kích thước n thành các bài toán con nhỏ hơn mà việc tìm lời giải của chúng là cùng một cách. Lời giải của bài toán đã cho được xây dựng từ các bài toán con này. Ta có thể nói vắn tắt ý tưởng chính của phương pháp này là : chia dữ liệu thành từng miền đủ nhỏ, giải bài toán trên các miền đã chia rồi tổng hợp kết quả lại.

Để có được mô tả chi tiết thuật toán chia để trị chúng ta cần phải xác định :

1. Kích thước tới hạn n_0 (Bài toán có kích thước nhỏ hơn n_0 sẽ không cần chia nhỏ)
2. Kích thước của mỗi bài toán con trong cách chia
3. Số lượng các bài toán con như vậy
4. Thuật toán tổng hợp lời giải của các bài toán con

Các phần xác định trong 2 và 3 phụ thuộc vào 4. Chia như thế nào để khi tổng hợp có hiệu quả (thường là tuyến tính)

3.3.2. Sơ đồ tổng quát thuật toán chia để trị

(Viết theo tựa Pascal):

Procedure D_and_C(n)

Begin

If $n = n_0$ Then

Giải bài toán một cách trực tiếp

Else

i. *Chia bài toán thành r bài toán con kích thước n/k*

ii. *For (Mỗi bài toán trong r bài toán con) Do*

D_and_C(n/k)

iii. Tổng hợp lời giải của r bài toán con để thu được lời giải của bài toán gốc

End;

(Viết theo tựa C#:)

Nếu gọi D&C (R) - R là miền dữ liệu - là hàm thể hiện cách giải bài toán theo phương pháp chia để trị thì ta có thể viết :

```
void D&C(R)
{       If ( R đủ nhỏ)
                giải bài toán;
        Else
                {       Chia R thành R1, . .., Rm ;
                        for (i = 1; i <=m; i++)
                                D&C(Ri);
                                Tổng hợp kết quả ;
                }
}
```

3.4. Bài toán tìm kiếm nhị phân

**** Bài toán :***

Cho mảng gồm n phần tử đã được sắp xếp tăng dần và một phần tử x. Tìm xem x có trong mảng hay không? Nếu có x trong mảng thì trả ra kết quả là 1, nếu không trả ra kết quả là 0.

Dùng thuật toán tìm kiếm nhị phân,

- ***Phân tích thuật toán :***

Số x cho trước

+ Hoặc là bằng phần tử nằm ở vị trí giữa mảng

+ Hoặc là nằm ở nửa bên trái (x < phần tử ở giữa mảng)

+ Hoặc là nằm ở nửa bên phải (x > phần tử ở giữa mảng)

Mô tả thuật toán:

Input: mảng a[1..n]

Output: + 1 nếu x thuộc a

+ 0 nếu x không thuộc a

Từ nhận xét đó ta có giải thuật sau:

```
Tknp (a, x, Đầu, Cuối) {
            If (Đầu > Cuối)
             return 0 ; {dãy trống}
            Else
            { Giữa = (Đầu + cuối) / 2;
             If (x == a[Giữa])
                Return 1;
            else
               if (x > a[Giữa])
                        Tknp(a, x, Giữa + 1, Cuối) ;
               else
                        Tknp(a, x, Đầu, Giữa - 1) ;
            }
```

**** Đánh giá độ phức tạp thời gian của thuật toán***

a) Trường hợp tốt nhất: Tương ứng với sự tìm được y trong lần so sánh đầu tiên, tức là x= a[giữa] (x nằm ở vị trí giữa mảng)

=> Ta cĩ : $T_{tốt}(n) = O(1)$

b) Trường hợp xấu nhất: Độ phức tạp là O(log n)

Thật vậy, Nếu gọi T(n) là độ phức tạp của thuật toán, thì sau khi kiểm tra điều kiện (x == a[giữa]) và sai thì gọi đệ qui thuật toán này với dữ liệu giảm nửa, nên thỏa mãn công thức truy hồi :

T(n) = 1 + T(n/2) với n>=2 và T(1) = 0.

Đánh giá ĐPTTT theo định lý thợ rút gọn

T(n) = 1.T(n/2)+1.n^0

Nhận xét: Với a=1, b=2, k=0 thì a=b^k=1 => theo ĐL thợ RG có T(n)=O(lgn)

3.5. Bài toán max và min

T(n)=2T(n/2)+c.n^0

Đánh giá ĐPTTT theo định lý thợ rút gọn

T(n)=2T(n/2)+c.n^0

Nhận xét: Với a=2, b=2, k=0 thì a=2>b^k=1 => theo ĐL thợ RG có

T(n)= O(n^ ($\log_2 2$))=O(n)

3.6. Thảo luận về chia để trị

**** Bài toán:***

Cho T[1..n] là một mảng n phần tử. Vấn đề đặt ra là sắp xếp các phần tử này theo thứ tự tăng. Chúng ta đã có thể giải quyết vấn đề này bằng các phương pháp *selection sort hay insertion sort hoặc là heapsort*

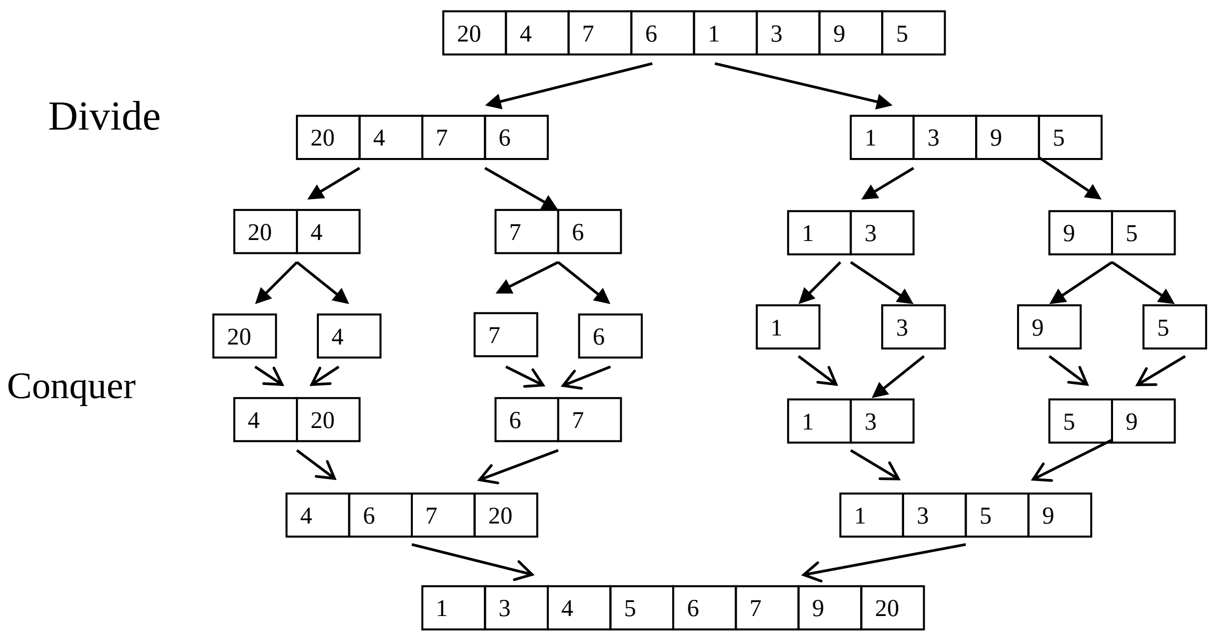
Như chúng ta đã biết thời gian dùng selection sort hay insertion sort để sắp xếp mảng T trong cả hai trường hợp: xấu nhất và trung bình đều vào cỡ n^2. Còn heapsort vào khoảng nlogn.

Có một số giải thuật đặc biệt cho bài toán này theo mô hình chia để trị đó là ***mergesort và quicksort, chúng ta sẽ lần lượt đi nghiên cứu chúng.***

- ***MergeSort***

Chia để trị tiếp cận tới bài toán này bằng việc tách mảng T thành hai phần mà kích thước của chúng sai khác nhau càng ít càng tốt, sắp xếp các phần này bằng cách gọi đệ qui và sau đó trộn chúng lại (chú ý duy trì tính thứ tự). Để làm được điều này chúng ta cần một giải thuật hiệu quả cho việc trộn hai mảng đã được sắp U và V thành một mảng mới T mà kích thước của mảng T bằng tổng kích thước của hai mảng U và V. Vấn đề này có thể thực hiện tốt hơn nếu ta thêm vào các ô nhớ có sẵn ở cuối của mảng U và V các giá trị cầm canh (giá trị lớn hơn tất cả các giá trị trong U và V, giả sử là)

Hình sau chỉ ra các bước của mergesort.



Hình 3.1

Mảng đã được sắp theo thứ thự tăng

Giải thuật sắp xếp này minh hoạ tất cả các khía cạnh của chia để trị. Khi số lượng các phần tử cần sắp là nhỏ thì ta thường sử dụng các giải thuật sắp xếp đơn giản.

Khi số phần tử đủ lớn thì ta chia mảng ra 2 phần, tiếp đến trị từng phần một và cuối cùng là kết hợp các lời giải.

*Thuật toán

```
Merge_Sort(A,L,R)
{
If(L>=R) return;
Mid= (L+R)/2;
Merge_Sort(A,L, Mid);
Merge_Sort(A,Mid+1,R);
Merge(A,L,Mid, R);
}
Procedure Merge(A,L,M, R)
{
int i=L, j=M+1, k=L;
while(i<=M && j<=R)
{
if(A[i]<A[j]){
        B[k]= A[i];
        i++;
        }
else{B[k]=A[j]; j++}
```

```
k++;
}
while(i<=M){B[k]=A[i]; i++; k++;}
while(j<=R){B[k]=A[j]; j++; k++;}
}
```

* Độ phức tạp thuật toán:

Bài toán chia thành các bước

Với mỗi lần merge thứ *i*, độ phức tạp bài toán là *O(n)*

Số lần merge là *O(log n)*

Thời gian tổng cộng: O(*n* log*n*)

Như vậy hiệu quả của mergesort tương tự heapsort. Trong thực tế sắp xếp trộn có thể nhanh hơn heapsort một ít nhưng nó cần nhiều hơn bộ nhớ cho các mảng trung gian U và V. Ta nhớ lại heapsort có thể sắp xếp tại chỗ (in-place), và cảm giác nó chỉ sử dụng một ít biến phụ mà thôi. Theo lý thuyết mergesort cũng có thể làm được như vậy tuy nhiên giá thành có tăng một chút ít.

- ***Quicksort:***

Giải thuật này được phát minh bởi Hoare, nó thường được hiểu như là tên gọi của nó - sắp xếp nhanh, hơn nữa nó cũng dựa theo nguyên tắc chia để trị. Không giống như mergesort nó quan tâm đến việc giải các bài toán con hơn là sự kết hợp giữa các lời giải của chúng. Bước đầu tiên của giải thuật này là chọn 1 vật trung tâm (pivot) từ các phần tử của mảng cần sắp. Tiếp đến vật trung tâm sẽ ngăn mảng này ra 2 phần: các phần tử lớn hơn vật trung tâm thì được chuyển về bên phải nó, ngược lại thì chuyển về bên trái. Sau đó mỗi phần của mảng được sắp xếp độc lập bằng cách gọi đệ qui giải thuật này. Cuối cùng mảng sẽ được sắp xếp xong. Để cân bằng kích thước của 2 mảng này ta có thể sử dụng phần tử ở giữa (median) như là vật trung tâm. Đáng tiếc là việc tìm phần ở giữa cũng mất 1 thời gian đáng kể. Để giải quyêt điều đó đơn giản là chúng ta sử dụng 1 phần tử tuỳ ý trong mảng cần sắp như là vật *trung tâm và hi vọng nó là tốt nhất có thể.*

Việc thiết kế giải thuật ngăn cách mảng bằng vật trung tâm với thời gian tuyến tính không phải là sự thách đố (có thể làm được). Tuy nhiên điều đó là cần thiết để so sánh với các giải thuật sắp xếp khác như là heapsort. Vấn đề đặt ra là mảng con T[i..j] cần được ngăn bởi vật trung tâm p=T[i]. Một cách làm có thể chấp nhận được là: Duyệt qua từng phần tử của của nó chỉ một lần nhưng bắt đầu từ hai phía (đầu và cuối mảng). Khi đó khởi tạo k=i; l=j+1, k tăng dần cho đến khi T[k] > p, l giảm dần cho đến khi T[l] (l. Tiếp đến hoán vị T[k] và T[l]. Quá trình này tiếp tục cho đến khi k (l. Cuối cùng đổi chỗ T[i] và T[l] cho nhau và lúc này ta xác định đúng vị trí của phần tử trung tâm.

* Mô tả thuật toán:

```
quicksort(L)
{
 if (length(L) < 2) return L
 else
 {
        pick some x in L  // x là phần tử chốt
        L1 = { y in L : y < x }
        L2 = { y in L : y > x }
        L3 = { y in L : y = x }
         quicksort(L1)
         quicksort(L2)
         return concatenation of L1, L3, and L2
 }
}
```

- Thuật toán

```
QuickSort(A, L, R)
{
P = Partition(A, L, R);
QuickSort(A, L,P-1);
QuickSort(A, P+1, R);
}

Partition(A, L, R)
{
I=L+1; J=R; P=L;
While(I<=J)
{
        While(i<= R && A[i]<A[P]) i++;
        While(j>=L && A[j]>A[P]) j--;
```

```
        If(I<J){
                Doi cho A[i] voi A[j];
                I++;
                J--;
        }
    }
    // Dua chot ve dung vi tri
    K= J;
    Doi cho A[p] voi A[j];
    Return K;
    }
```

Hình vẽ sau cho thấy sự làm việc của pivot và quicksort.

3	1	4	1	5	9	2	6	5	3	5	8	9

3	1	4	1	5	9	2	6	5	3	5	8	9

3	1	$\underline{4}$	1	5	9	2	6	5	$\bar{3}$	5	8	9

3	1	3	1	5	9	2	6	5	4	5	8	9

3	1	3	1	$\underline{5}$	9	$\bar{2}$	6	5	4	5	8	9

3	1	3	1	2	9	5	6	5	4	5	8	9

3	1	3	1	$\bar{2}$	$\underline{9}$	5	6	5	4	5	8	9

2	1	3	1	3	9	5	6	5	4	5	8	9

1	1	2	3	3	4	5	5	5	6	8	9	9

Hình 3.2

Quicksort không hiệu quả nếu sử dụng việc gọi đệ quy của các bài toán con mà không chú ý đến sự cân bằng kích thước của chúng. Tình huống xấu nhất là khi T đã được sắp trước mà gọi quicksort và thời gian dùng quicksort để sắp là (O(n^2).

Gọi t(n) là thời gian trung bình dùng quicksort để sắp mảng n phần tử T[1..n]. l là giá trị trả về khi gọi pivot(T[1..n],l). Theo pivot() thì l nằm giữa 1 (n và xác suất là 1/n. Thời gian để tìm vật trung tâm g(n) là tuyến tính. Thời gian để dùng đệ qui để sắp xếp hai mảng con kích thước (l-1) và (n-l) tương ứng là t(n-1) và t(n-l). Như vậy với n đủ lớn ta có:

$$t(n) = \frac{1}{n}\sum_{l=1}^{n}(g(n) + t(n-1) + t(n-l))$$

Hay rõ ràng hơn ta chọn n0 là giá trị đủ lớn để sử dụng công thức trên. Nghĩa là nếu n < n0 thì ta dùng sắp xếp chèn. Khi đó gọi d là hằng số sao cho g(n) ≤ dn với $n>n_0$.

Ta có t(n) ≤ dn + $\frac{1}{n}\sum_{l=1}^{n}(g(n) + t(n-1) + t(n-l))$ với n>n0

≤ dn + $\frac{2}{n}\sum_{k=0}^{n-1}t(k)$ với $n>n_0$ (2.4)

Với công thức kiểu như 2.4 quả là khó phân tích độ phức tạp. Ta dự đoán nó tương tự mergesort và hi vọng nó là như vậy tức là vào cỡ O(nlogn).

Thật vậy ta có định lý sau:

Định lý: *Quicksort cần nlogn thời gian để sắp xếp n phần tử trong trường hợp trung bình.*

Chứng minh

Gọi t(n) là thời gian cần thiết để sắp n phần tử trong trường hợp trung bình.

a, n_0 là các hằng số giống như công thức 2.4

Ta chứng minh t(n) = cnlogn với mọi n ≥ 2. với c là hằng số.

Dùng phương pháp qui nạp để c/m:

- Với mọi n nguyên dương: (2 ≤ n ≤ n0)

  Dễ thấy t(n) ≤ cnlogn

  Chọn c ≥ $\frac{t(n)}{n\log n}, \forall n;$ 2 ≤ n ≤ n0) (2.5)

- Bước qui nạp

  Giả thiết rằng: t(k) ≤ cklogk với mọi 2≤ k < n

  ta chỉ ra c sao cho t(n) ≤ cnlogn

  Lấy a = t(0) +t(1)

Từ 7.2 ta có

t(n) = dn + $\frac{2}{n}\sum_{k=0}^{n-1} t(k)$

$$t(n) = dn + \frac{2}{n}\left(t(0) + t(1) + \sum_{k=0}^{n-1} t(k)\right)$$

Theo giả thiết qui nạp : t(k) = cklogk

$$\Rightarrow t(n) \le dn + \frac{2a}{n} + \frac{2}{n}\left(\sum_{k=2}^{n-1} ck\log k\right)$$

$$\le dn + \frac{2a}{n} + \frac{2c}{n}\int_{k=2}^{n} x\log x dx$$

$$= dn + \frac{2a}{n} + \frac{2c}{n}\left(\frac{x^2\log x}{2} - \frac{x^2}{4}\right)\Bigg|_{x=2}^{n}$$

$$\le dn + \frac{2a}{n} + \frac{2c}{n}\left(\frac{n^2\log n}{2} - \frac{n^2}{4}\right)$$

$$= dn + \frac{2a}{n} + cn\log n - \frac{cn}{2}$$

$$= cn\log n - \left(\frac{c}{2} - d - \frac{2a}{n^2}\right)n .$$

t(n) = cnlogn với điều kiện là $\left(\frac{c}{2} - d - \frac{2a}{n^2}\right) \geq 0$, hay c ≥ 2d + 4a/n2

Từ đó chúng ta chỉ xem xét với những n > n_0 thoả mãn:

$$c = 2d + \frac{4a}{(n_0+1)^2} \qquad (2.6)$$

Kết hợp cả 2 điều kiện trong (2.5) và (2.6) ta có

$$c = \max\left(2d + \frac{4(t(0)+t(1))}{(n_0+1)^2}, \max\left\{\frac{t(n)}{n\log n}, 2 \leq n \leq n_0\right\}\right) \qquad (2.7)$$

Hay ta có t(n) ≤ cnlogn với mọi n ≥ 2, và như vậy định lý được chứng minh.

Như vậy quicksort có thể sắp xếp 1 mảng n phần tử khác nhau trong trường hợp trung bình là O(nlogn). Câu hỏi đặt ra là liệu có thể sửa đổi quicksort để nó sắp xếp với thời gian O(nlogn) trong trương hợp xấu nhất hay không. Câu trả lời là có thể!!!. Tuy nhiên nếu việc tìm phần tử ở giữa của T[i..j] là tuyến tính và lấy nó làm vật trung tâm (pivot) (Finding the median) thì quicksort cũng cần O(n2) để sắp xếp n phần tử trong trường hợp xấu nhất (khi tất cả các phần tử của mảng cần sắp là bằng nhau).

3.6.2. Thuật toán nhân 2 ma trận

**** Bài toán :***

Cho hai ma trận A, B với kích thước n*n, ta có ma trận C chứa kết quả của phép nhân hai ma trận A và B. Thuật toán nhân ma trận cổ điển như công thức dưới đây:

$$C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}$$

- ***Phân tích thuật toán***

Với mảng một chiều (kích thước n phần tử), ma trận C được tính trong thời gian (n), giả sử rằng phép cộng vô hướng và phép nhân là các phép tính cơ bản (có thời gian tính là hằng số). Với mảng hai chiều (kích thước n*n) thì thời gian để tính phép nhân ma trận AB là (n).

Đến cuối những năm 1960, Strassen đưa ra một giải pháp cải tiến thuật toán trên, nó có tính đột phá trong lịch sử của thuật toán chia để trị, thậm chí gây ngạc nhiên không kém thuật toán nhân số nguyên lớn được phát minh ở thập kỷ trước. ý tưởng cơ bản của thuật toán Strassen cũng tương tự như thuật toán trên. Ứng dụng thiết kế chia để trị, mỗi ma trận A, B, C ta chia thành 4 ma trận con và biểu diễn tích 2 ma trận AxB = C theo các ma trận con đó:

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

Trong đó :

$C_{11} = A_{11}B_{11} + A_{12}B_{21}$

$C_{12} = A_{11}B_{12} + A_{12}B_{22}$

$C_{21} = A_{21}B_{11} + A_{22}B_{21}$

$C_{22} = A_{21}B_{12} + A_{22}B_{22}$

Nếu theo cách nhân thông thường, thì cách chia để trị này dẫn đến công hưc truy hồi : $T(n) = 8T(n/2) + O(n^2)$. Đáng tiếc là kết quả này cho lời giải T(n) thuộc $O(n^3)$

Nhưng theo khám phá của Strassen, chỉ cần 7 phép nhân đệ qui n/2x n/2 ma trận và $O(n^2)$ phép cộng trừ vô hướng theo công thức truy hồi :

$T(n) = 7T(n/2) + 18 (n/2)^2$ € $O(n^{\lg 7}) = O(n^{2.81})$

Cụ thể, để nhân 2 ma trận vuông cấp 2, theo Strassen chỉ cần 7 phép nhân và 18 phép cộng (trừ) các sô. Để tính :

$$\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

- Đầu tiên tính 7 tích :

$m_1 = (a_{12} - a_{22})(b_{21} + b_{22})$

$m_2 = (a_{11} + a_{22})(b_{11} + b_{22})$

$m_3 = (a_{11} - a_{21})(b_{11} + b_{12})$

$m_4 = (a_{11} + a_{12})\, b_{22}$

$m_5 = a_{11}(b_{12} - b_{22})$

$m_6 = a_{22}(b_{21} - b_{11})$

$m_7 = (a_{21} + a_{22})\, b_{11}$

- sau đó tính c_{ij} theo công thức :

$c_{11} = m_1 + m_2 - m_4 + m_6$

$c_{12} = m_4 + m_5$

$c_{21} = m_6 + m_7$

$c_{22} = m_2 - m_3 + m_5 - m_7$

Mô tả thuật toán:

```
strass(a, b, c, n)
if ( n == 2 )       nhan2(a,b,c);
else
 {
        tach(a,a11,a12,a21,a22,n);
        tach(b,b11,b12,b21,b22,n);
        tach(c,c11,c12,c21,c22,n);
```

```
strass(a11,b11,d1,n/2);
strass(a12,b21,d2,n/2);
 cong(d1,d2,c11,n/2);

strass(a11,b12,d1,n/2);
strass(a12,b22,d2,n/2);
 cong(d1,d2,c12,n/2);



 strass(a21,b11,d1,n/2);
strass(a22,b21,d2,n/2);
cong(d1,d2,c21,n/2);

strass(a21,b12,d1,n/2);
strass(a22,b22,d2,n/2);
cong(d1,d2,c22,n/2);

Hop(c11,c12,c21,c22,c,n);
}
```

Cùng với phát minh của Strassen, có một số nhà nghiên cứu cố gắng tìm kiếm thuật toán để xác định được hằng số ω, khi đó độ phức tạp tính toán phép nhân hai ma trận kích thước n*n là $O(n^w)$. Để thực hiện được điều này, việc đầu tiên phải tiến hành là nhân hai ma trận kích thước 2*2 với 6 phép nhân cơ bản. Nhưng vào năm 1971 Hopcroft và Kerr đã chứng minh điều này là không thể vì phép nhân hai ma trận không có tính chất giao hoán. Việc tiếp theo phải thực hiện là tìm cách nào để nhân hai ma trận 3*3 với nhiều nhất chỉ 21 phép nhân cơ bản. Nếu thực hiện được việc này có thể sử dụng thuật toán này để suy ra thuật toán đệ quy nhân hai ma trận n*n với thời gian là $\Theta(n^{\log_3 21})$, nhanh hơn thuật toán của Strassen vì $\log_3 21 < \log_2 7$. Không may mắn là điều này không thể thực hiện. Trong suốt một thập kỷ trước khi Pan phát hiện ra cách để nhân hai ma trân kích thước 70*70 vớI 143640 phép nhân cơ bản – so sánh với 343000 phép nhân nếu sử dụng thuật toán cổ điển và quả thực là $\log_{70} 143640$ bé hơn một chút so vớI lg7. Phát hiện này được gọi là cuộc chiến tranh của số thập phân (decimal war). Nhiều thuật toán, mà trong đó hiệu suất tiệm cận cao, được tìm ra sau đó. Ví dụ như ta đã biết cuốI năm 1979, phép nhân ma trận có thời gian tính toán là $O(n^{2,521813})$. Hãy tưởng tượng rằng, ngay sau đó tháng 1 năm 1980 thờI gian tính của phép nhân ma trận là $O(n^{2,521801})$. Biểu thức tiệm cận thời gian tính toán tồi nhất của thuật toán nhân ma trận kích thước n*n được Coppersmith và Winograd phát minh ra năm 1986 là $O(n^{2,376})$. Tại vì các hằng số liên quan bị ẩn nên không một thuật toán nào được tìm ra sau thuật toán của Strassen được nghiên cứu và sử dụng.

4.6.3. Bài toán hoán đổi

**** Bài toán:***

a[1..n] là một mảng với phần gồm n phần tử. Ta cần chuyển m phần tử đầu tiên của mảng với phần còn lại của mảng (n-m phân tử) mà không dùng một mảng phụ.

Chẳng n, với n = 8 a[8] = (1, 2, 3, 4,5, 6, 7, 8)

Nếu m = 3, thì kết quả là : (4, 5, 6, 7, 8, 1, 2, 3)

Nếu m = 5, thì kết quả là : (6, 7, 8, 1, 2, 3, 4, 5)

Nếu m = 4, thì kết quả là : (5, 6, 7, 8, 1, 2, 3, 4)

****Phân tích thuật toán :***

Nếu m = n- m : Hoán đổi các phần tử của 2 nửa mảng có độ độ dài bằng nhau :

Nếu m <> n-m :

- Nếu m < n - m : hoán đổi m phần tử đầu với m phân tử cuối của phần còn lại. Sau đó trong mảng a[1..n-m] ta chỉ cần hoán đổi m phần tử đầu với phần còn lại.

- Nếu m > n - m : hoán đổi n-m phần tử đầu tiên với n-m phần tử sau. Sau đó trong mảng a[n-m+1. . n] ta hoán đổi n-m phần tử cuối mảng với các phần tử của phần đầu.

Như vậy, bằng cách áp dụng phương pháp chia để trị, ta chia bài toán thành 2 bài toán con:

- Bài toán thứ nhất là hoán đổi hai mảng con có độ dài bằng nhau, cụ thể là phần tử đầu và cuối của mảng cho nhau bằng cách đổi chỗ từng cặp phần tử tương ứng.

- Bài toán thứ hai cùng dạng với bài toán đã cho nhưng kích thước nhỏ hơn, nên có thể gọi thuật toán đệ qui để giải và quá trình gọi đệ qui sẽ dừng khi đạt tới sự hoán đổi 2 phần có độ dài bằng nhau

**** Thuật toán :***

// Hoán đổi m phần tử

Input : a[1..n], m. (m <=n)

Output : a[1..n] với tính chất m phần tử đầu mảng a (mảng nhập) nằm cuối mảng kết quả, n-m phần tử cuối mảng nhập nằm ở đầu mảng kết quả.

Thuật toán:

```
HOANDOI(A, L, R, m)

{

n= L-R+1; // So luong phan tu co trong day A[L], A[L+1], …, A[R]

If(n-m==m){

 for(i=0; i< m;i++)
```

```
        Đổi chỗ A[i+L] ↔A[n-i-L+1];
}
if(m > n-m){
for(i=0; i< n-m; i++ )
        Đổi chỗ A[i+L] ↔A[n-i-L+1];
HOANDOI(A, n-m+1, R, m )
}
if(m < n-m){
for(i=0; i< m; i++ )
        Đổi chỗ A[i+L] ↔A[R-i];
HOANDOI(A,1, n-m, m );
}
}
```

**** Độ phức tạp thuật toán:***

Kí hiệu : T(i, j) là số phần tử cần đổi chỗ để hoán đổi dãy i phần tử và dãy j phần tử, ta có công thức truy hồi sau:

$$T(i,j) = \begin{cases} i;\ \textbf{\textit{nếu i}} = \boldsymbol{j} \\ j + T(i-j, j);\ \textbf{\textit{nếu i}} > \boldsymbol{j} \\ i + T(i, j-i);\ \textbf{\textit{nếu i}} < \boldsymbol{j} \end{cases}$$

=> T(i,j) = i + j – UCLN(i,j) (Ước chung lớn nhất của i, j)

BÀI 5. CƠ BẢN VỀ THUẬT TOÁN QUY HOẠCH ĐỘNG

5.1. Sơ đồ chung của thuật toán

Quy hoạch động có những nét giống như phương pháp "Chia để trị", nó đòi hỏi việc chia bài toán thành những bài toán con kích thước nhỏ hơn. Như chúng ta đã thấy trong chương trước, phương pháp chia để trị chia bài toán cần giải ra thành các bài toán con độc lập, sau đó các bài toán con này được giải một cách đệ quy, và cuối cùng tổng hợp các lời giải của các bài toán con ta thu được lời giải của bài toán đặt ra. Điểm khác cơ bản của quy hoạch động với phương pháp chia để trị là các bài toán con là không độc lập với nhau, nghĩa là các bài toán con cùng có chung các bài toán con nhỏ hơn. Trong tình huống đó, phương pháp chia để trị sẽ tỏ ra không hiệu quả, khi nó phải lặp đi lặp lại việc giải các bài toán con chung đó. Quy hoạch động sẽ giải một bài toán con một lần và lời giải của các bài toán con sẽ được ghi nhận, để thoát khỏi việc giải lại bài toán con mỗi khi ta đòi hỏi lời giải của nó.

Quy hoạch động thường được áp dụng để giải các bài toán tối ưu. Trong các bài toán tối ưu, ta có một tập các lời giải, mà mỗi lời giải như vậy được gán với một giá trị số. Ta cần tìm lời giải với giá trị số tối ưu (nhỏ nhất hoặc lớn nhất). Lời giải như vậy ta sẽ gọi là lời giải tối ưu.

Việc phát triển giải thuật dựa trên quy hoạch động có thể chia làm 3 giai đoạn:

- Phân rã: Chia bài toán cần giải thành những bài toán con nhỏ hơn có cùng dạng với bài toán ban đầu sao cho bài toán con kích thước nhỏ nhất có thể giải một cách trực tiếp. Bản thân bài toán xuất phát có thể coi là bài toán con có kích thước lớn nhất trong họ các bài toán con này.

- Ghi nhận lời giải: Lưu trữ lời giải của các bài toán con vào một bảng. Việc làm này là cần thiết vì lời giải của các bài toán con thường được sử dụng lại rất nhiều lần, và điều đó nâng cao hiệu quả của giải thuật do không phải giải lặp lại cùng một bài toán nhiều lần.

- Tổng hợp lời giải: Lần lượt từ lời giải của các bài toán con kích thước nhỏ hơn tìm cách xây dựng lời giải của bài toán kích thước lớn hơn, cho đến khi thu được lời giải của bài toán xuất phát (là bài toán con có kích thước lớn nhất). Kỹ thuật giải các bài toán con của quy hoạch động là quá trình đi từ dưới lên (bottom – up) là điểm khác quan trọng với phương pháp chia để trị, trong đó các bài toán con được trị một cách đệ quy (top – down).

Yêu cầu quan trọng nhất trong việc thiết kế thuật toán nhờ quy hoạch động là thực hiện khâu phân rã, tức là xác định được cấu trúc của bài toán con. Việc phân rã cần được tiến hành sao cho không những bài toán con kích thước nhỏ nhất có thể giải được một cách trực tiếp mà còn có thể dễ dàng việc thực hiện tổng hợp lời giải.

Không phải lúc nào việc áp dụng phương pháp quy hoạch động đối với bài toán tối ưu hoá cũng dẫn đến thuật toán hiệu quả. Có hai tính chất quan trọng mà một bài toán tối ưu cần phải thoả mãn để có thể áp dụng quy hoạch động để giải nó là:

- Cấu trúc con tối ưu: Tính chất này còn được gọi là tiêu chuẩn tối ưu và có thể phát biểu như sau: Để giải đực bài toán đặt ra một cách tối ưu, mỗi bài toán con cũng phải được giải một cách tối ưu. Mặc dù sự kiện này có vẻ là hiển nhiên, nhưng nó thường không được thoả mãn do các bài toán con là giao nhau. Điều đó dẫn đến là một lời giải có thể là “kém tối ưu hơn” trong một bài toán con này nhưng lại có thể là lời giải tốt trong một bài toán con khác.

- Số lượng các bài toán con phải không quá lớn. Rất nhiều các bài toán NP – khó có thể giải được nhờ quy hoạch động, nhưng việc làm này là không hiệu quả do số lượng các bài toán con tăng theo hàm mũ. Một đòi hỏi quan trọng đối với quy hoạch động là tổng số các bài toán con cần giải là không quá lớn, cùng lắm phải bị chặn bởi một đa thức của kích thước dữ liệu vào.

5.2. Tập con độc lập lớn nhất trên cây

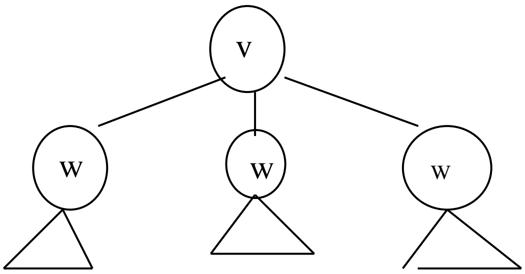
Để phát biểu bài toán ta cần nhắc lại một số khái niệm. Giả sử G = (V,E) là đơn đồ thị vô hướng với trọng số trên đỉnh c(), V. Một tập con các đỉnh của đồ thị được gọi là tập độc lập, nếu như hai đỉnh bất kỳ trong U là không kề nhau trên G.

Nếu U là tập độc lập, thì ta gọi trọng số của U là tổng trọng số của các đỉnh trong nó. Ta sẽ gọi tập độc lập với trọng số lớn nhất là tập độc lập lớn nhất. Bài toán tập độc lập lớn nhất trên đồ thị là một bài toán khó. Tuy nhiên, khi đồ thị G là cây bài toán này có thể giải hiệu quả bởi thuật toán quy hoạch động.

Bài toán phát biểu như sau: Cho cây T = (V,E) với trọng số trên các đỉnh c(), V. Hãy tìm tập độc lập lớn nhất của T.

Dựng cây T có gốc tại đỉnh r, và duyệt cây theo thứ tự sau (postorder). Xét đỉnh tuỳ ý với k con w1, w2, …, wk. Ta có thể tạo tập độc lập của cây con gốc tại theo hai cách, phụ thuộc vào việc ta có chọn đỉnh vào tập độc lập hay không:

- Nếu ta không chọn vào tập độc lập, thì ta có thể kết hợp các tập độc lập của các cây con gốc tại w1, w2, …, wk để tạo tập độc lập gốc tại, bởi vì không có cạnh nối giữa các cây con này.
- Còn nếu ta chọn vào tập độc lập thì ta chỉ có thể sử dụng các tập độc lập không chứa gốc của các cây con tương ứng với w1, w2, …, wk, do và bất kỳ con wi nào của nó không cùng chọn vào tập độc lập



Do đó, với mỗi đỉnh thuật toán phải tính các thông tin sau:

1. big(u) = trọng lượng lớn nhất của tập độc lập của cây con có gốc tại u.
2. bibnotroot(v) = trọng lượng lớn nhất của tập độc lập không chứa của cây con có gốc tại v.

Tại đỉnh v, thuật toán sẽ gọi đệ quy tính big(wi) và bignotroot(wi) với mỗi cây con gốc tại các con w1, w2, …, wk của. Sau đó tính bignotroot(v) và big(v) sử dụng công thức đệ quy tương ứng với hai tình huống mô tả ở trên:

$$bign\text{otroot}(\text{v}) = \sum_{i=1}^{k} big(w_i)$$

$$big(v) = \max\{bignotroot(v), c(v) + \sum_{i=1}^{k} bignotroot(w_i)\}$$

Nếu là lá thì bignotroot(v) = 0 và big(v) = c(v).

Từ các phân tích trên dễ dàng xây dựng thuật toán tính big(v), V với thời gian O(n), trong đó n =.

Ta xét cách thực hiện đệ quy của thuật toán. Rõ ràng trọng số của tập độc lập lớn nhất tại đỉnh sẽ hoặc là bằng trọng lượng của tất cả các tập độc lập của các cây con gốc tại w1, w2, …, wk hoặc bằng tổng trọng lượng của và trọng lượng của các cây con gốc tại các đỉnh là cháu của. Từ đó ta có thuật toán đệ quy sau:

```
function MaxISTree(r);
(* Tìm tập độc lập lớn nhất của cây con gốc tại r   *)
(* Con(r) - danh sách các con của root                *)
(* Cháu(r) - danh sách các cháu của root          *)
begin
   if Con(r) = then return c(r) (* r là lá *)
   else
       begin
           bignotroot: = 0;
           for w ∈ Con(r) do
           bignotroot: = bignotroot + MaxISTree(w);
           bigr: = c(r);
           for u ∈ Chau(r) do
           bigr: = bigr + MaxISTree(u);
           return max(bignotroot, bigr);
      end;
end;
```

Lệnh gọi MaxISTree(root) (root là gốc của cây T) sẽ thực hiện thuật toán. Tất nhiên thủ tục đệ qui này là không hiệu quả. Do ở đây chỉ có O(n) bài toán con cần giải, ta có thể viết lại nó dưới dạng thủ tục đệ qui có nhớ để có được thuật toán với thời gian tính O(n).

PHẦN ĐỌC THÊM

5.3. Bài toán nhân ma trận

Như đã biết, tính của ma trận A = (a_{ik}) kích thước p x q với ma trận B = (b_{kj}) kích thước q x r là ma trận C = (c_{ij}) kích thước p x r với các phần tử được tính theo công thức:

$$c_{ij} = \sum_{k=1}^{q} a_{ik} b_{ki}, 1 \le i \le p, 1 \le j \le q.$$

Chúng ta có thể sử dụng đoạn chương trình sau đây để tính tích của hai ma trận A,B:

```
for i =1 -> p
  for j =1 -> r
{
   c [i,j] = 0
                    for k = 1 -> q do
                              c[i,j] =c[i,j] +a[i,k] *b[k,j];
}
```

Rõ ràng, đoạn chương trình trên đòi hỏi thực hiện tât cả pqr phép nhân để tính tích của hai ma trận.

Giả sử ta phải tích tích của nhiều hơn là hai ma trận. Do phép nhân ma trận có tính kết hợp, ta có thể tích tích của các ma trận.

$M = M_1 M_2 \ldots Mn$

Theo nhiều cách khác nhau, chẳng hạn

$M =(\ldots((M_1 M_2)M_3)\ldots Mn)$

$= (M1(M_2(M_3\ldots(Mn_{-1}Mn)\ldots)))$

$=(\ldots((M_1M_2)(M_3M_4))\ldots),v.v\ldots$

Mặt khác, do tích ma trận không có tính chất giao hoán, nên ta không được thay đổi thứ tự của các ma trận trong biểu thức đã cho.

Mỗi cách tính tích các ma trận đó cho đòi hỏi một thời gian tính khác nhau. Để đánh giá hiệu quả của các phương pháp chúng ta đếm số phép nhân cần phải thực hiện. Trong đoạn chương trình trên, ta thấy để tính tích của hai ma trận ta còn phải thực hiện cùng một số như vậy các phép cộng và một số phép gán và tính chỉ số, vì thế, số lượng phép nhân là một chỉ số đánh giá khá chính xác hiệu quả của phương pháp.

Ví dụ: Tính tích M = ABCD của bốn ma trận, trong đó A có kích thước 13 x 5, B có kích thước 5 x 89, C có kích thước 89 x 3 và D có kích thước 3 x 34. Sử dụng cách tính

M = ((AB)C)D),

Ta phải thực hiện lần lượt tính

AB 5785 phép nhân

(AB)C3271 phép nhân

((AB)C)D 1326 phép nhân

Và tổng cộng là 10582 phép nhân

Tất cả có 5 phương pháp khác nhau để tính tích ABCD:

1. ((AB)C)D 10582
2. (AB)(CD) 54201
3. (A(BC))D 2856
4. A((BC)D) 4055
5. A(B(CD)) 26418

Phương pháp hiệu quả nhất (phương pháp 3) đòi hỏi khối lượng phép nhân ít hơn gần 19 lần so với phương pháp tồi nhất (phương pháp 5).

Để tìm phương pháp hiệu quả nhất, chúng ta có thể liệt kê tất cả các cách điền dấu ngoặc vào biểu thức tích ma trận đã cho và tính số lượng phép nhân đòi hỏi theo mỗi cách. Ký hiệu T (n) là số cách điền các dấu ngoặc vào biểu thức tích của n ma trận. Giả sử ta định đặt dấu ngoặc phân tách đầu tiên vào giữa ma trận thứ i và ma trận thứ (i + 1) trong biểu thức tích, tức là:

$$M = (M_1 M_2 \ldots Mi)(Mi_{+1} Mi_{+2} \ldots Mn)$$

Khi đó có T(i) cách đặt dấu ngoặc cho thừa số thứ nhất ($M_1 M_2 \ldots Mi$) và T(n-i) cách đặt dấu ngoặc cho thừa số thứ hai ($Mi_{+1} Mi_{+2} \ldots Mn$) và từ đó T(i)T(n-i) cách tính biểu thức ($M_1 M_2 \ldots Mi$)($Mi_{+1} Mi_{+2} \ldots Mn$). Do i có thể nhận bất cứ giá trị nào trong khoảng từ 1 đến n-1, suy ra ta có công thức truy hồi sau để tính T(n):

Kết hợp với điều kiện đầu hiển nhiên T(1) = 1, ta có thể tính các giá trị của T(n) với mọi n. Bảng dưới đây cho một số giá trị của T(n).

n	1	2	3	4	5	10	15
T(n)	1	1	2	5	14	4862	26744 40

Giá trị của T(n) được gọi là số Catalan. Công thức sau đây cho phép tính T(n) qua hệ số tổ hợp:

$$T(n) = \frac{1}{n} C_{2n-2}^{n-1}, \ n \geq 2$$

Từ đó T(n) = T(n) = 4^{nn2}. Như vậy, phương pháp duyệt toàn bộ không thể sử dụng để tìm cách tính hiệu quả biểu thức tính của n ma trận, khi n lớn.

Bây giờ, ta xét cách áp dụng quy hoạch động để giải bài toán đặt ra.

* Phân rã (Xác định cấu trúc con tối ưu).

Bước đầu tiên phải thực hiện khi muốn áp dụng quy hoạch động để giải bài toán đặt ra là tiến hành phân rã bài toán hay phát hiện cấu trúc con tối ưu. Nhận thấy rằng: Nếu cách tính tối ưu tích của n ma trận đòi hỏi dặt dấu ngoặc tách đầu tiên giữa ma trận thứ i và thứ (i+1) của biểu thức tích, thì khi đó cả hai tích con ($M_1 M_2$ … Mi) và (Mi_{+1} Mi_{+2} … Mn) cũng phải được tính một cách tối ưu. Khi đó số phép nhân cần phải thực hiện để nhân dãy ma trận sẽ bằng tổng số phép nhân cần thực hiện để nhân hai dãy con (($M_1 M_2$ … Mi) và (Mi_{+1} Mi_{+2} … Mn) cộng với số phép nhân cần thực hiện để nhân hai ma trận kết quả tương ứng với hai dãy con này. Vì vậy để xác định cách thực hiện nhân tối ưu ta cần giải quyết hai vấn đề sau:

- Cần đặt dấu ngoặc phân tách đầu tiên vào vị trí nào (xác định i);
- Thực hiện việc tính tối ưu hai tích con ($M_1 M_2$ … Mi) và (Mi_{+1} Mi_{+2} … Mn) bằng cách nào.

Việc tính mỗi tích con rõ ràng có dạng giống như bài toán ban đầu, vì thế có thể giải một cách đệ quy bằng cách áp dụng cách giải như đối với dãy xuất phát. Vấn đề thứ nhất có thể được giải bằng cách xét tất cả các giá trị có thể của i. Như vậy, bài toán nhân dãy ma trận thoả mãn đòi hỏi về cấu trúc con tối ưu: Để tìm cách tính tối ưu việc nhân dãy ma trận ($M_1 M_{2}$. . M_n) chúng ta có thể sử dụng cách tính tối ưu của hai tích con ($M_1 M_2$ … Mi) và (Mi_{+1} Mi_{+2} … Mn). Nói cách khác, những bài toán con phải được giải một cách tối ưu cũng như bài toán ban đầu. Phân tích này cho phép ta sử dụng quy hoạch động để giải bài toán đặt ra. Xét họ các bài toán: Tìm m_{ij} là số phép nhân ít nhất cần thực hiện để tính tích

$(M_{i+1} M_{i+2} \dots Mj)$, $1 \le i \le j \le n$

Lời giải cần tìm sẽ là m_{1n}

* Tổng hợp lời giải.

Giả sử kích thước của các ma trận được cho bởi véc tơ d[0 … n], trong đó ma trận Mi có kích thước $d_{i-1}xd_i$, i = 1, 2, 3, … n. Ta sẽ xây dựng bảng giá trị mi j lần lượt theo từng đường chéo của nó, trong đó đường chéo thứ s chứa các phần tử mi j với chỉ số thoả mãn j - i = s. Khi đó, đường chéo s = 0 sẽ chứa các phần tử mi_j (i = 1, 2, … n) tương ứng với tích có một phần tử Mi. Do đó, $mi_j = 0$, i = 1, 2, … n. Đường chéo s = 1 chứa các phần tử m_{ij+1} tương ứng với tích M_iM_{i+1}, do ở đây không có sự lựa chọn nào khác, nên ta phải thực hiện $d_{i-1}d_id_{i+1}$ phép nhân. Giả sử s > 1, khi đó đường chéo thứ s chứa các phần tử mij+s tương ứng với tích Mi Mi+1 … Mi+s. Bây giờ ta có thể lựa chọn việc đặt dấu ngoặc tách đầu tiên sau một trong số các ma trận Mi, Mi+1, .., Mi+s-1. Nếu đặt dấu ngoặc đầu tiên sau Mk, i ≤ k < i+s, ta cần thực hiện mik phép nhân để tính thừa số thứ nhất, mk+1,i+s phép nhân để tính thừa số thứ hai, và cuối cùng là di-1dkdi+s phép nhân để tính tích của hai ma trận thừa số để thu được ma trận kết quả. Để tìm cách tính tối ưu, ta cần chọn cách đặt dấu ngoặc tách đòi hỏi ít phép nhân nhất.

Như vậy, để tính bảng giá trị mi j ta có thể sử dụng quy tắc sau đây:

s = 0; $mi_j = 0$ i = 1, 2, …, n

s = 1; $mi_{j+1} = d_{i-1}d_id_{i+1}$, i = 1, 2, …, n - 1

1 < s < n; $mi_{j+s} = \min\{mik + mk_{+1,\,i+s} + d_{i-1}dkd_{i+s} : 1 \le k < i+s\}$, i = 1, 2, …, n – s.

Lưu ý rằng, để dễ theo dõi ta viết cả công thức cho trường hợp s = 1, mà dễ thấy là công thức cho trường hợp tổng quát vẫn đúng cho s = 1.

Ví dụ 2: Tìm cách tính tối ưu cho tích của bốn ma trận cho trong ví dụ 1.

Ta có d = (13, 5, 89, 3, 34). Với s = 1, m12 = 5785, m23 = 1335 và m34 = 9078.

Tiếp theo, với s = 2 ta thu được

$m_{13} = \min(m_{11} + m_{23} + 13 \times 5 \times 3, m_{12} + m_{33} + 13 \times 89 \times 3)$
$= \min(1530, 9256) = 1530$
$m_{24} = \min(m_{22} + m_{34} + 5 \times 89 \times 34, m_{23} + m_{44} + 5 \times 3 \times 34)$
$= \min(24208, 1845) = 1845$

Cuối cùng với s = 3 ta có

$m_{14} = \min(m_{11} + m_{24} + 13 \times 5 \times 34),\ \{k = 1\}$
$m_{12} + m_{34} + 13 \times 89 \times 34, \{k = 2\}$
$m_{13} + m_{44} + 13 \times 3 \times 34,\ \{k = 3\}$
$= \min(4055, 54201, 2856) = 2856.$

Bảng giá trị m được cho trong hình vẽ dưới đây

	j=1	2	3	4	
i=1	0	5785	1530	2856	
2		0	1335	1845	s=3
3			0	9078	s=2
4				0	s=1
					s=0

Để tìm lời giải tối ưu, ta sử dụng bảng hi j ghi nhận cách đặt dấu ngoặc tách đầu tiên cho giá trị mi j.

Ví dụ 3: Các giá trị của hi j theo ví dụ 1 được cho trong bảng dưới đây:

	j=1	2	3	4	
i=1	1	2	1	3	
2		2	3	3	s=3
3			3	4	s=2
4				4	s=1
					s=0

Ta có số phép nhân cần thực hiện là m14 = 2856. Dấu ngoặc đầu tiên cần đặt sau vị trí h14 = 3, tức là M = (ABC)D. Ta tìm cách đặt dấu ngoặc đầu tiên để có m13 tương ứng với tích ABC. Ta có h13 = 1, tức là tích ABC được tính tối ưu theo cách: ABC = A(BC). Từ đó suy ra, lời giải tối ưu là: M = (A(BC))D.

Bây giờ, ta tính số phép toán cần thực hiện theo thuật toán vừa trình bày. Với mỗi s > 0, có n s phần tử trên đường chéo cần tính, để tính mỗi phần tử đó ta cần so sánh s giá trị số tương ứng với các giá trị có thể của k. Từ đó suy ra số phép toán cần thực hiện theo thuật toán là cỡ

$$\sum_{s=1}^{n-1}(n-s)s = n\sum_{s=1}^{n-1}s - \sum_{s=1}^{n-1}s^2$$
$$= n^2(n-1)/2 - n(n-1)(2n-1)/6$$
$$= (n^3-n)/6$$
$$= 0(n^3)$$

Thuật toán trình bày có thể mô tả trong hai thủ tục sau:

procedure Matrix-Chain(d,n)

{m[i,j] - chi phí tối ưu thực hiện nhân dãy Mi. . . Mj;

h[i,j] - ghi nhận vị trí đặt dấu ngoặc đầu tiên trong cách thực hiện nhân dãy Mi. . . Mj}

```
begin
for i: = 1 to n do m[i,j]: = 0;            //khởi tạo
for s: = 1 to n do                         // s = chỉ số của đường chéo
        for i: = 1 to n - s do
        begin
                j: = i + s - 1; m[i,j] = +?;
                for k: = i to j - 1 do
                begin
                        q: = m[i,k] + m[k+1,j] + d[i-1]*d[k]*d[j];
                        if(q<m[i,j]) then
                        begin
                                m[i,j] = q; h[i,j] = k;
                        end;
                end;
```

```
        end;
end;
```

Thủ tục đệ quy sau đây sử dụng mảng ghi nhận h để đưa ra trình tự nhân tối ưu.

```
procedure Mult(i,j);
begin
        if(i<j) then
        begin
                k = h[i,j];
                X = Mult(i,k);          { X = M[i] /. . . M[k]          }
                Y = Mult(k+1,j);        { Y = M[k+1]. . . M[j]      }
                return X*Y;             { Nhân ma trận X và Y       }
        end
                else
                        return M[i];
end;
```

BÀI 6. CÁC BÀI TOÁN SỬ DỤNG THUẬT TOÁN QUY HOẠCH ĐỘNG

6.1 Bài toán dãy con lớn nhât

Ta đã trình bày thuật toán chia để trị để giả bài toán dãy con lớn nhất với thời gian tính 0 (n log n). Bây giờ ta xét thuật toán quy hoạch động để giải bài toán này. Để đơn giản ta chỉ xét cách tính tổng của dãy con lớn nhất.

Phân rã. Gọi si là tổng của dãy con lớn nhất trong dãy

a1, a2, …., ai,

i = 1,2,…, n. Rõ ràng sn là giá trị cần tìm.

Tổng hợp lời giải. Trước hết, ta có s1 = a1. Bây giờ giả sử i > 1 và sk là đã biết với k = 1,2,…, i - 1. Ta cần tính si là tổng của dãy con lớn nhất của dãy con lớn nhất của dãy a1, a2, …, ai-1, ai.

Rõ ràng dãy con lớn nhất của dãy này hoặc là có chứa phần tử ai hoặc là không chứa phần tử ai, vì thế chỉ có thể là một trong hai dãy sau đây:

- Dãy con lớn nhất của dãy a1, a2, …, ai-1.
- Dãy con lớn nhất của dãy a1, a2, …, ai kết thúc tại ai.

Từ đó suy ra

si = max {si-1,ei },

Trong đó ei là tổng của dãy con lớn nhất của dãy a1, a2, …, ai kết thúc tại ai.

Lưu ý rằng để tính ei, i = 1, 2, …, n, ta cũng có thể sử dụng công thức đệ quy sau:

e1 = a1;

ei = max {ai, ei-1 + ai }, i > 1.

Từ đó, ta có thuật toán sau để giải bài toán đặt ra:

procedure M axsub(a);

```
begin
   smax: = a[1];                    (* smax - tổng của dãy con lớn nhất *)
   maxendhere: = a[1];
   imax: = 1;              (* imax - vị trí kết thúc của dãy con lớn nhất *)
```

```
    for i: = 2 to n do
    begin
            u: = maxendhere + a[i];
            v: = a[i];
            if (u > v) then maxendhere = u else maxendhere = v;
            if (maxendhere > smax) then
            begin
                    smax: = maxendhere;
                    imax: = i;
            end;
    end;
end;
```

Dễ thấy thuật toán Maxsub có thời gian tính là O(n).

6.2. Bài toán dãy con chung dài nhất

Ta gọi dãy con của một dãy cho trước là dãy thu được từ dãy đã cho bằng việc loại bỏ một số phần tử. Một cách hình thức, giả sử cho dãy X = <x1, x2, …, xm>, dãy Z = <z1, z2, …, zk> được gọi là dãy con của dãy X nếu tìm được dãy các chỉ số 1? i1 < i2 < … < ik ? n sao cho zi = xi j, j = 1, 2, …, k. Chẳng hạn dãy Z = < B, C, D, B> là dãy con của dãy X = <A, A, B, C, B, C, D, A, B, D, A, B> với dãy chỉ số là <3, 4, 7, 9>.

Cho hai dãy X và Y ta nói dãy Z là dãy con chung của hai dãy X và Y nếu Z là dãy con của cả hai dãy này. Ví dụ, nếu X = <A, B, C, D, E, F, G> và Y = <C, C, E, D, E, G, F> thì dãy Z = <C, D, F> là dãy con chung của hai dãy X và Y, còn dãy <B, F, G> không là dãy con chung của chúng. Dãy <C, D, F> không là dãy con chung dài nhất vì nó có độ dài 3 (số phần tử trong dãy), trong khi đó dãy <C, D, E, G> là dãy con chung của X và Y có độ dài 4. Dãy <C, D, E, G> là dãy con chung dài nhất vì không tìm được dãy con chung có độ dài 5.

Bài toán dãy con chung dài nhất được phát biểu như sau: Cho hai dãy X = <x1,x2,…,xm> và Y = <y1,y2,…,yn>. Cần tìm dãy con chung dài nhất của hai dãy X và Y.

Thuật toán trực tiếp để giải bài toán đặt ra là: Duyệt tất cả các dãy con của dãy X và kiểm tra xem mỗi dãy như vậy có là dãy con của dãy Y, và giữ lại dãy con dài nhất. Mỗi dãy con của X tương ứng với dãy chỉ số <i1,i2, …, ik> là tập con k phần tử của tập chỉ số {1, 2, …, m}, vì thế có tất cả 2m dãy con của X.Như vậy thuật toán trực tiếp đòi hỏi thời gian hàm mũ và không thể ứng dụng được trên thực tế. Ta xét áp dụng quy hoạch động để xây dựng thuật toán giải bài toán này.

Phân rã. Với mỗi 0 <= i <= m và 0 <= j <= n xét bài toán C(i,j); tính C[i,j] là độ dài của dãy con chung dài nhất của hai dãy.

Xi=<x1,x2,…,xi>

và

yi=<y1,y2,…yj>

Như vậy ta đã phân bài toán cần giải ra thành (m+1)x(n+1) bài toán con. Bản thân bài toán xuất phát là bài toán con có kích thước lớn nhất C(m,n).

Tổng hợp lời giải. Rõ ràng

c[o,j]=0,j =0,1,…,n và c[i,0]=0,i=0,1,…,m.

Giả sử i>0,j>0 ta cần tính c[i,j] là độ dài của dãy con chung lớn nhất của hai dãy Xi và Yj có hai tình huống:

Nếu Xi =Yj thì dãy con chung dài nhất của Xi vàYj sẽ thu được bằng việc bổ sung Xi vào dãy con chung dài nhất của hai dãy Xi-1và Yj-1

Nếu Xi #Yj thì dãy con chung dài nhất của Xi và Yj sẽ là dãy con dài nhất trong hai dãy con chung dài nhất của (Xi và Yj-1) và của (Xi-1 và Yj). Từ đó ta có công thức sau để tính C[i,j].

$$c[i,j]\begin{cases}0, & \text{nếu i=0 hoặc j=0}\\ c[i-1,j-1]+1 \quad , & \text{nếu i,j >0 và xi=yj}\\ \max\{c[i,j-1],c[i-1,j]\} & \text{nếu i,j >0 và xi } yi\end{cases}$$

Thuật toán tìm dãy con chung dài nhất có thể mô tả nh- sau.

```
Procedure LCS(X,Y);
    begin
            for i :=1 to m do c[i,0]:=0;
     forj: =1 to n do c[0,j:=0;
  for i: =1 to m do
   for j: = 1to n do
    if xi = yi then
```

```
  begin
   c[i,j]:=c[i-1,j-1]+1;
   b[i,j]:=/;
  end
 else
  if c [i-1,j]≥c[i,j-1] then
  begin
   c[i,j]:=c[i-1,j];
   b[i,j]:=↑;
  end
  else
  begin
   c[i,j]:=c[i,j-1];
   b[i,j]:=←;
  end;
 end;
```

Trong thủ tục mô tả ở trên ta sử dụng biến b[i,j] để ghi nhận tình huống tối ưu khi tính giá trị c[i,j]. Sử dụng biến này ta có thể đưa ra dãy con chung dài nhất của hai dãy X và Y nhờ thủ tục sau đây:

```
Procedure Print LCS (b,X,i,j);
 begin
  if(i=0)or (j=0) then return;
  if b[i,j]=/ then
   begin
    print LCS (b,X,i-1,j-1);
    print xi;      (* Đ- a ra phân tử xi *)
   end
   else
   if b[i,j]=↑then
    PrintLCS (b,X,i-1,j)
  else
    Print LCS (b,X,i,j-1);
 end;
```

Dễ dàng đánh giá đ- ợc thời gian tính của thuật toán LCS là 0(mn).

PHẦN ĐỌC THÊM

6.3.Thuật toán Floyd- tìm đường đi ngắn nhất giữa các cặp đỉnh

1. Bài toán

Cho G=(V, E), là một đơn đồ thị có hướng, Trong đó V là tập các đỉnh, E là tập các cung. Tìm đường đi ngắn nhất giữa các cặp cạnh của đồ thị.

2. Ý tưởng

Thuật toán Floyd được thiết kế theo phương pháp quy hoạch động. Nguyên lý tối ưu được vận dụng cho bài toán này là :

« Nếu k là đỉnh nằm trên đường đi ngắn nhất từ i đến k và từ k đến j cũng phải ngắn nhất »

3. Thiết kế

Đồ thị được biểu diễn bởi ma trận kề các trọng số của cung a= $(a_{ij})_{nxn}$

$$\forall i,j \in \{1,..,n\} : a_{ij} = \begin{cases} Trọng\ số\ (i,j); (i,j) \in E \\ 0; i = j \\ \infty; (i,j) \notin E \end{cases}$$

Ta ký hiệu :

- Ma trận trọng số đường đi ngắn nhất giữa các cặp đỉnh : d= $d_{i\,j}$

$d_{i\,j}$ Trọng số của đường đi ngắn nhất từ i đến j

p_{ij} : Đường đi ngắn nhất từ i đến j có đi qua đỉnh trung gian hay không

$$\begin{cases} p_{ij} = 0; \text{ đường đi ngắn nhất từ i đến j không có đi qua đỉnh trung gian } p_{ij}. \\ p_{ij} \neq 0; \text{ đường đi ngắn nhất từ i đến j đi qua đỉnh trung gian } p_{ij}. \end{cases}$$

- Ở bước k :

+ Ký hiệu ma trận d là d^k cho biết chiều dài nhỏ nhất của đường đi từ i đến j.

+ Ký hiệu ma trận p là P^k cho biết đường đi ngắn nhất từ i đến j có đi qua đỉnh trung gian thuộc tập đỉnh {1, …, k}.

Input: a

Output: d, p

Mô tả:

Bước 1: khởi tạo d,p:

d = a; (d^0_{ij})

$p_{i\,j}$ =0

Bước 2:

Kiểm tra mỗi cặp đỉnh I, j có hay không có đường đi từ I đến j đi qua đỉnh trung gian 1, mà có trọng số nhỏ hơn bước 1? Trọng số của đường đi đó là:

$d^1_{ij} = \mathrm{Min}\{ d^0_{ij}, d^0_{i1} + d^0_{1j} \}$

Nếu $d^1_{ij} = d^0_{i1} + d^0_{1j}$ thì $P^1_{\ ij} = 1$, tức là đường đi tương ứng đi qua đỉnh 1

Bước 3:

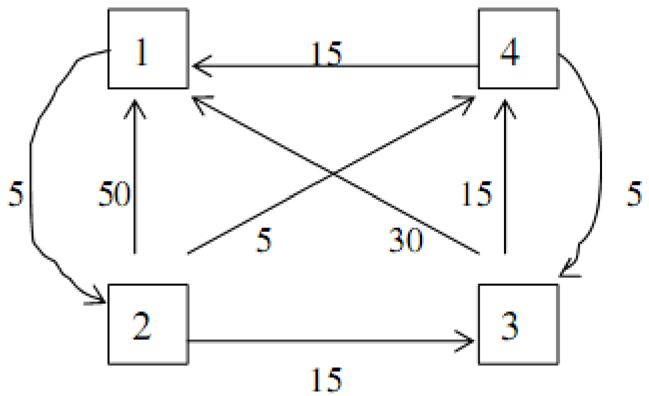
Kiểm tra mỗi cặp đỉnh I, j có hay không có đường đi từ I đến j đi qua đỉnh trung gian 2, mà có trọng số nhỏ hơn bước 2? Trọng số của đường đi đó là:

$d^2_{ij} = \text{Min}\{ d^1_{ij}, d^1_{i2} + d^1_{2j} \}$

Nếu $d^2_{ij} = d^1_{i2} + d^1_{2j}$ thì $P^2_{ij} = 1$, tức là đường đi tương ứng đi qua đỉnh 2

… Bước n Cứ tiếp tục như vậy thuật toán kết thúc sau bước n,ma trận d xác định trọng số đường đi ngắn nhất giữa 2 đỉnh bất kỳ i, j. Ma trận p cho biết đường đi ngắn nhất từ i đến j có đi qua đỉnh trung gian p_{ij} hay không.

Minh họa : Tìn đường đi ngắn nhất giữa các cặp đỉnh của đồ thị



Hoạt động của thuật toán Floyd :

b1		1	2	3	4
d^1	1	0	5	∞	∞
	2	50	0	15	5
	3	30	35	0	15
	4	15	20	5	0

p^1		1	2	3	4
	1	0	0	0	0
	2	0	0	0	0
	3	0	1	0	0
	4	0	1	0	0

b2		1	2	3	4
d^2	1	0	5	20	10
	2	50	0	15	5
	3	30	35	0	15
	4	15	20	5	0

p^2		1	2	3	4
	1	0	0	2	2
	2	0	0	0	0
	3	0	1	0	0
	4	0	1	0	0

b3		1	2	3	4
d^3	1	0	5	20	10
	2	45	0	15	5
	3	30	35	0	15
	4	15	20	5	0

p^3		1	2	3	4
	1	0	0	2	2
	2	3	0	0	0
	3	0	1	0	0
	4	0	1	0	0

b4		1	2	3	4
d^4 =	1	0	5	15	10
d	2	20	0	10	5
	3	30	35	0	15
	4	15	20	5	0

		1	2	3	4
p^4 =	1	0	0	4	2
p	2	4	0	4	0
	3	0	1	0	0
	4	0	1	0	0

4. Thuật toán

```
void floyd()
{
        int i, j, k;
        // Khoi dong ma tran d va p
        for (i = 1; i<= n; i++)
                for (j = 1; j<= n; j++)
                {
                        d[i][j] = a[i][j];
                        p[i][j] = 0;
                }
for (k = 1; k <= n; k++)  // Tính ma trận d và p ở bước lặp k
        for (i = 1; i <= n; i++)
                if ( d[i][k] > 0 && d[i][k] < vc )
                        for (j = 1; j<= n; j++)
                                if ( d[k][j] > 0 && d[k][j] < vc )
                                        if (d[i][k] + d[k][j] < d[i][j] )
                                        {
                                                d[i][j] = d[i][k] + d[k][j];
                                                p[i][j] = k;
                                        }
}
```

5. **Độ phức tạp của thuật toán**

$T(n)= O(n^3)$

6.3. Bài tập về quy hoạch động

Bài 1:Viết chương trình cài đặt thuật toán tìm kiếm nhị phân

- Bài toán : Cho mảng a[1..n] được sắp xếp theo thứ tự không giảm và x. Tìm x trong mảng a, nếu có trả về giá trị 1, nếu không có trả về giá trị 0

- Phân tích thuật toán :

Số x cho trước

+ Hoặc là bằng phần tử nằm ở vị trí giữa mảng a

+ Hoặc là nằm ở nửa bên trái (x < phần tử ở giữa mảng a)

+ Hoặc là nằm ở nửa bên phải (x > phần tử ở giữa mảng xa)

- Cài đặt thuật toán:

```
int     tknp(int a[max],int x,int l, int r)
{
        int mid;
        if( l > r) return 0;
        mid = (l+r)/2
        if ( x == a[mid] ) return 1;
        if ( x > a[mid] ) return tknp(a,x,mid+1,r);
        return tknp(a,x,l,mid-1);
```

- Mở rộng:

Sửa đổi đoạn chương trình trên với yêu cầu trả về vị trí tìm được của x trong mảng a, nếu không tìm thấy trả về giá trị -1

Bài 2. Viết chương trình cài đặt thuật toán mảng con lớn nhất

- Bài toán:

Tìm giá trị Min, Max trong đoạn a[l..r] của mảng a[1..n].

- Phân tích thuật toán:

+ Tại mỗi bước, chia đôi đoạn cần tìm rồi tìm Min, Max của từng đoạn, sau đó tổng hợp lại kết quả.

+ Nếu đoạn chia chỉ có 1 phần tử thì Min =Max và bằng phần tử đó

Ví dụ:

i	1	2	3	4	5	6	7	8
a[i]	10	1	5	0	9	3	15	19

MinMax(a,2,6,Min,Max) cho Min = 0, Max = 9 trong đoạn a[2..6]

- Cài đặt thuật toán:

Input : a[l..r], (l <= r)

Output: Min = Min(a[l]..a[r])

Max = Max(a[l]..a[r])

```
void MinMax(int a[.], int  l, int  r, int &Min, int &Max )
{
        int Min1,Min2,Max1,Max2;
        if (l == r )
        {
                Min = a[l];
                Max= a[l];
        }
        else
        {
                MinMax(a,l,(l+r)/2 , Min1, Max1);
                MinMax(a,(l+r) /2 + 1,r, Min2, Max2);
                if (Min1 < Min2)
                        Min = Min1;
                else
                        Min = Min2;
                if (Max1 > Max2)
                        Max = Max1;
                else
                        Max = Max2;
        }
}
```

Bài 3. Viết chương trình cài đặt thuật toán sắp xếp QuickSort

- Bài toán:

Dùng thuật toán QuickSort (QS) để sắp xếp các giá trị trong một mảng các số theo thứ tự,chẳng hạn tăng dần.

- Phân tích thuật toán:

Chọn ngẫu nhiên một phần tử x.

Duyệt dãy từ bên trái (theo chỉ số i) trong khi còn a_i < x.

Duyệt dãy từ bên phải (theo chỉ số j) trong khi còn a_j > x.

Đổi chỗ ai và aj nếu hai phía chưa vượt qua nhau.

.. . tiếp tục qúa trình duyệt và đổi chỗ như trên trong khi hai phía còn chưa vượt qua nhau (tức là còn có i = j).

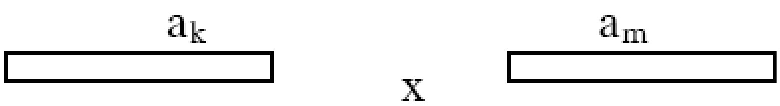
Kết qủa phân hoạch dãy thành 3 phần :

+ a_k <= x với k = 1,. ., j (Dãy con thấp);

+ a_m >= x với với m = i,. ..,n (Dãy con cao);

+ a_h = x với h = j+1, …, i+1

(Vì thế phương pháp này còn gọi là phương pháp sắp xếp bằng phân hoạch)



Tiếp tục phân hoạch cho phần trái (dãy con thấp nhỏ hơn x), cho phần phải (lớn hơn x). . . cho đến khi các phân hoạch chỉ còn lại một phần tử, là sắp xếp xong.

Thuật toán thể hiện ý tưởng đệ qui và cách thiết kế chia để trị.

- Thuật toán QuickSort

Input : a[1..n]

Output : a[1..n] không giảm.

Thuật toán

```
QS(A, L, R)
{
If(L>=R) return;
Else
{
I=L; j=R;
While(I<=J)
{
While(A[i]<A[L] )i++;
```

```
While(A[j]>A[L] )j--;
If(I<= J){ Doi cho A[i]<--> A[j];
I++; J--;
}
DoiCho(A[L], A[j]);
QS(A, L, J-1);
QS(A, J+1, R);
}
}
}
```

BÀI 7. CƠ BẢN VỀ THUẬT TOÁN THAM LAM

7.1 Đặc trưng của chiến lược tham lam

Phương pháp tham lam là kỹ thuật thiết kế thường được dùng để giải các bài toán tối ưu. Phương pháp được tiến hành trong nhiều bước. Tại mỗi bước, theo một chọn lựa nào đó (xác định bằng một hàm chọn), sẽ tìm một lời giải tối ưu cho bài toán nhỏ tương ứng. Lời giải của bài toán được bổ sung dần từng bước từ lời giải của các bài toán con. Lời giải được xây dựng như thế có chắc là lời giải tối ưu của bài toán ?

Các lời giải theo phương pháp tham lam thường chỉ là chấp nhận được theo điều kiện nào đó, chưa chắc là tối ưu.

Cho trước một tập A gồm n đối tượng, ta cần phải chọn một tập con S của A. Với một tập con S được chọn ra thỏa mãn các yêu cầu của bài toán, ta gọi là một nghiệm chấp nhận được. Một hàm mục tiêu gắn mỗi nghiệm chấp nhận được với một giá trị. Nghiệm tối ưu là nghiệm chấp nhận được mà tại đó hàm mục tiêu đạt giá trị nhỏ nhất (lớn nhất).

Đặc trưng tham lam của phương pháp thể hiện bởi : trong mỗi bước việc xử lí sẽ tuân theo một sự chọn lựa trước, không kể đến tình trạng không tốt có thể xảy ra khi thực hiện lựa chọn lúc đầu.

7.2. Sơ đồ chung của thuật toán

**** Đặc điểm chung của thuật toán tham lam:***

- Mục đích xây dựng bài toán giải nhiều lớp bài toán khác nhau, đưa ra quyết định dựa ngay vào thuật toán đang có, và trong tương lai sẽ không xem xét lại quyết định trong quá khứ. - > thuật toán dễ đề xuất, thời gian tính nhanh nhưng thường không cho kết quả đúng.
- Lời giải cần tìm có thể mô tả như là bộ gồm hữu hạn các thành phần thoả mãn điều kiện nhất định, ta phải giải quyết bài toán một cách tối ưu -> hàm mục tiêu
- Để xây dựng lời giải ta có một tập các ứng cử viên

- Xuất phát từ lời giải rỗng, thực hiện việc xây dựng lời giải từng bước, mỗi bước sẽ lựa chọn trong tập ứng cử viên để bổ xung vào lời giải hiện có.
- Xây dựng được một hàm nhận biết được tính chấp nhận được của lời giải hiện có -> Hàm Solution(S) -> Kiểm tra thoả mãn điều kiện.
- Một hàm quan trọng nữa : Select(C) cho phép tại mỗi bước của thuật toán lựa chọn ứng cử viên có triển vọng nhất để bổ xung vào lời giải hiện có -> dựa trên căn cứ vào ảnh hoởng của nó vào hàm mục tiêu, thực tế là ứng cử viên đó phải giúp chúng ta phát triển tiếp tục bài toán.
- Xây dựng hàm nhận biết tính chấp nhận được của ứng cử viên được lựa chọn, để có thể quyết định bổ xung ứng cử viên được lựa chọn bởi hàm Select vào lời giải -> Feasible(S ẩ x).

Sơ đồ thuật toán

```
* input A[1..n]
* output S //lời giải;
greedy (A,n)
  S = 0;
while ( A <> 0)
  {
                x= Chọn(A); A = A-{x}
     if( S U {x} chấp nhận được )
       S = S U {x};
               Return S;
  }
```

7.3. Bài toán người du lịch

**** Bài toán***

Một người du lịch muốn tham quan n thành phố T1,.., Tn. Xuất phát từ một thành phố nào đó, người du lịch muốn đi qua tất cả các thành phố còn lại, mỗi thành phố đi qua đúng 1 lần rồi quay trở lại thành phố xuất phát.

Gọi Cij là chi phí đi từ thành phố Ti đến Tj. Hãy tìm một hành trình thỏa yêu cầu bài toán sao cho chi phí là nhỏ nhất.

**** Phân tích, thiết kế thuật toán:***

Đây là bài toán tìm chu trình có trọng số nhỏ nhất trong một đơn đồ thị có hướng có trọng số. Thuật toán tham lam cho bài toán là chọn thành phố có chi phí nhỏ nhất tính từ thành phố hiện thời đến các thành phố chưa qua

Input C= (Cij)

output TOUR // Hành trình tối ưu,

Mô tả :

COST;//Chi ph í tương ứng

TOUR := 0; COST := 0; v := u; // Khởi tạo

Mọi k := 1 -> n ://Thăm tất cả các thành phố

// Chọn cạnh kề)

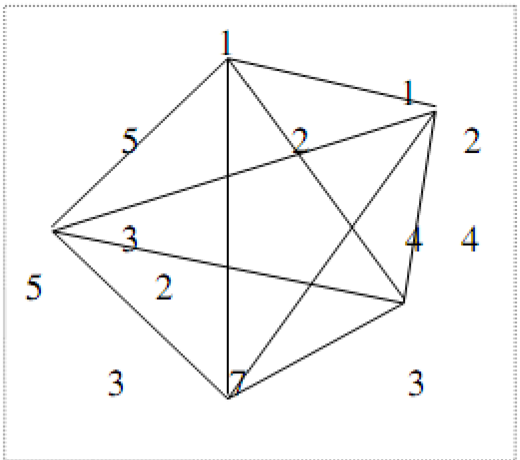
- Chọn <v, w> là đoạn nối 2 thành phố có chi phí nhỏ nhất tính từ thành phố v đến các thành phố chưa qua.

- TOUR := TOUR + <v, w>; //Cập nhật lời giải

- COST := COST + Cvw ; //Cập nhật chi phí

// Chuyến đi hoàn thành TOUR := TOUR + <v, u>; COST := COST + C_{vw}

Minh họa:



$$C = \begin{bmatrix} 0 & 1 & 2 & 7 & 5 \\ 1 & 0 & 4 & 4 & 3 \\ 2 & 4 & 0 & 1 & 2 \\ 7 & 4 & 1 & 0 & 3 \\ 5 & 3 & 2 & 3 & 0 \end{bmatrix}$$

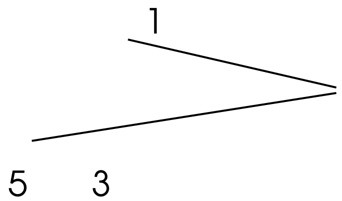
1.

TOUR := 0; COST := 0; u := 1;

=> w = 2;

2

2. TOUR := <1,2>; COST := 1; u := 2;

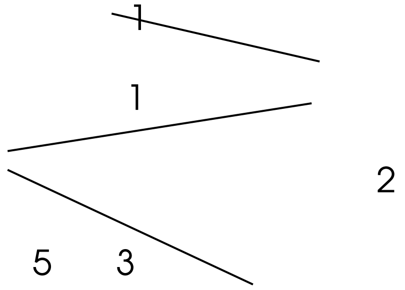


=> w = 5; 1 2

3. TOUR := {<1,2>, <2,5>}

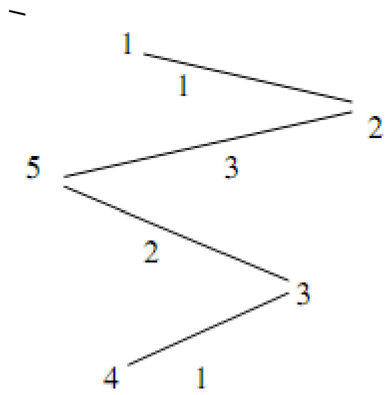
COST := 4; u := 5;

=> w = 3;

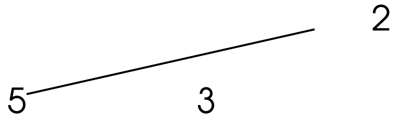


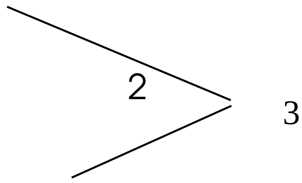
4. TOUR := {<1,2>, <2,5>,<5,3>} 2 3

COST := 6; u := 3;



=> w = 4; 1





5. TOUR := {<1,2>, <2,5>,<5,3>,<3,4>}

COST := 7; u = 1;

TOUR := {<1,2>, <2,5>,<5,3>,<3,4>,<4,1>}

COST := 14

**** Độ phức tạp thuật toán***

Thao tác chọn đỉnh thích hợp trong n đỉnh được tổ chức bằng một vòng lặp để duyệt. Nên chi phí cho thuật toán xác định bởi 2 vòng lặp lồng nhau, nên T(n) € O (n^2).

**** Cài đặt thuật toán***

```
 int GTS (mat a, int n, int TOUR[max], int Ddau)
 {
int             v,      //Dinh dang xet
k,                      //Duyet qua n dinh de chon
w;                      //Dinh duoc chon trong moi buoc
int             mini;  //Chon min cac canh(cung) trong moi buoc int COST;
                //Trong so nho nhat cua chu trinh
int daxet[max];             //Danh dau cac dinh da duoc su dung
for(k = 1; k <= n; k++)
daxet[k] = 0; //Chua dinh nao duoc xet
COST = 0;               //Luc dau, gia tri COST == 0
int i;          // Bien dem, dem tim du n dinh thi dung
v = Ddau;               //Chon dinh xuat phat la 1
i = 1;
TOUR[i] = v; //Dua v vao chu trinh
 daxet[v] = 1; //Dinh v da duoc xet
while(i < n)
  {
    mini = VC;
for (k = 1; k <= n; k++)
      if(!daxet[k])
if(mini > a[v][k])
```

```
 {
          mini = a[v][k];
   w = k;
 }
v = w;
i++;
TOUR[i] = v;
 daxet[v] = 1;
     COST += mini;
}
COST += a[v][Ddau];
return COST;
}
```

Bài 8. BÀI TẬP VÀ THẢO LUẬN TỔNG KẾT MÔN HỌC

8.1. Thuật toán Chia để trị

Bài toán MinMax

Input: Nhập vào danh sách n số ngẫu nhiên

Output: Số lớn nhấn Max và số nhỏ nhất Min của dãy

BinarySearch

Input: Nhập vào danh sách n số ngẫu nhiên. Số x cần tìm

Output: Kết quả trả về vt, nếu a[vt]=x; vt=-1 nếu ko có phần tử nào của dãy bằng x

So sánh BinarySearch với 1 thuật toán tìm kiếm tuần tự (Sequence Search)

Thuật toán sắp xếp Mergersort

Input: Nhập vào danh sách n số ngẫu nhiên

Output: Dãy số đã sắp xếp tang dần

Thuật toán sắp xếp QuickSort

Input: Nhập vào danh sách n số ngẫu nhiên

Output: Dãy số đã sắp xếp tang dần

So sánh QuickSort với 1 thuật toán sắp xếp trong (Insertion/Selection/Buble sort)

8.2. Quy hoạch động

Dáy con chung dài nhất

Input: Nhập 2 xâu XX, YY.

Output: Hãy tìm xâu con của XX và của YY có độ dài lớn nhất. Biết xâu con của một xâu thu được khi xóa một số kí tự thuộc xâu đó (hoặc không xóa kí tự nào).

Dáy con có trọng lượng lớn nhất

Input: Nhập dãy

Output: Hãy con có tổng lớn nhất

Trình tự nhân dãy ma trận tối ưu

Input: Nhập số ma trận và kích thước dãy ma trận

Output: trình tự nhân tối ưu

8.3 Tham

Bài toán đổi tiền

Input: Nhập vào 1 danh sách n cặp số nguyên tương ứng là mệnh giá mi ($) và số tờ tiền ci sẵn có trong 1 cây ATM của ngân hàng AI. Số tiền khách cần rút N$

Output: đưa ra một phương án trả tiền sao cho trả đủ n đồng cho khách và số tờ giấy bạc phải trả là ít nhất.

So sánh với thuật toán khác như vét cạn, QHĐ sẽ được khuyến khích

Bài toán lập lịch

Input: Nhập vào thông tin của n hội nghị đăng ký trong cùng 1 ngày sử dụng 1 phòng hội thảo. Mỗi hội nghị gồm: Tên hội nghị ni, Thời gian bắt đầu si, Thời gian kết thúc ti. (7<=si<ti<=22)

Output: Yêu cầu bố trí phòng họp sao cho phục vụ được nhiều cuộc họp nhất, biết mỗi thời điểm chỉ tổ chức 1 hội nghị, hội nghị này kết thúc, hội nghị khác mới bắt đầu.

Bài toán cái túi

Input: Nhập vào dãy n đồ vật. Mỗi đồ gồm Tên ni, Giá trị vi, trọng lượng wi. Trọng được túi chịu được M.

Output: Danh sách các đồ vật chọn được từ n đồ vật sao cho tổng giá trị lớn nhất, tổng trọng lượng đồ vật không vượt quá M.

Bài toán sắp công việc Cho n thiết bị (pi)1<= i <= n và m công việc (wi)1<=i <= m. Các thiết bị có thể làm việc đồng thời và làm việc nào cũng được. Mỗi việc đã làm ở thiết bị nào thì làm đến cùng. Thời gian làm công việc wi là ti, i €{1,..,m}. Cần xây dựng một lịch biểu là thứ tự thực hiện các công việc sao cho tổng thời gian hoàn thành là nhanh nhất.

BÀI 9. THỰC HÀNH ĐỘ PHỨC TẠP THUẬT TOÁN

- *Mục tiêu*

Lựa chọn thuật toán tốt để tiết kiện được thời gian, không gian nhớ tốt nhất cho mỗi bài toán

A. Bài tập mẫu

Tính giá trị đa thức

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots\ldots + a_1 x + a_0 \text{, với } x = x_0$$

Gợi ý

Xét thuật toán 1:

Tính giá trị từng hạng tử của đa thức, rồi cộng dồn lại

1. Với i = 1 đến n tính $a_i x_0^i$
2. Tính `p += a[i] *` $a_i x_0^i$`;`

Xét thuật toán 1 cải tiến: dung biến trung gian tg để lưu x_0^i

```
tg = tg * x;
p += a[i] * tg;
```

Xét Thuật toán 2 (thuật toán Hoocne): Biểu diễn lại đa thức

Đa thức P(x) có thể viết dưới dạng:

$$P(x) = (\ldots((a_n x + a_{n-1})x)\ldots)x + a_0.$$

Chẳng hạn với n = 3.

$$P(x) = a_3 x^3 + a_2 x^2 + a_1 x + a_0 = (((a_3 x + a_2)x + a_1) + a_0)$$

Xét Thuật toán 3:

1. $P := a_n$
2. Với i=1 đến n : $P := Px_0 + a_{n-i}$
3. Gán kết quả $P(x_0)= P$.

Chương trình mẫu

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Text;
```

```
namespace DaThuc
{
  class DaThuc
  {
    int[] a;
    public void Nhap(out int[] a)
    {
      Random r = new Random();
      Console.Write("Ban da thuc n=");
      int n = int.Parse(Console.ReadLine());
      a = new int[n + 1];
      for (int i = 0; i <= n; i++)
      {
        a[i] = r.Next(-10, 10);
      }
    }
    public void HienThi(int[] a)
    {
      Console.Write("\nDa thuc nhu sau:\nP=");
      for (int i = a.Length - 1; i > 0; i--)
      {
        Console.Write("{0}*x^{1}+", a[i], i);
      }
      Console.WriteLine(a[0]);
    }
    public double TinhDaThuc1(int[] a, double x)
    {
      double p = a[0];
      for (int i = 1; i < a.Length; i++)
      {
        p += a[i] * Math.Pow(x, i);
      }
      return p;

    }
    public double TinhDaThuc2(int[] a, double x)
    {
      double p = a[0], tg = 1;

      for (int i = 1; i < a.Length; i++)
      {
        tg = tg * x;
        p += a[i] * tg;
      }
      return p;
    }
    public double TinhDaThucHoocNe(int[] a, double x)
    {
      double p = a[a.Length - 1];
      for (int i = a.Length - 1; i > 0; i--)
        p = p * x + a[i];
```

```
      return p;
    }
    public void Test()
    {
      Nhap(out a);
      HienThi(a);
      Stopwatch st1 = new Stopwatch();
      double x = 1;
      st1.Start();
      Console.WriteLine("\n GTDT=" + TinhDaThuc1(a, x));
      st1.Stop();

      Stopwatch st2 = new Stopwatch();
      st2.Start();
      Console.WriteLine("\n GTDT=" + TinhDaThuc1(a, x));
      st2.Stop();
      Stopwatch st3 = new Stopwatch();
      st3.Start();
      Console.WriteLine("\n GTDT=" + TinhDaThuc1(a, x));
      st3.Stop();
      Console.WriteLine("Voi n={0} TG chay 3 TT lan luot la:\n
{1}\t{2}\t{3}", a.Length - 1, st1.Elapsed, st2.Elapsed, st3.Elapsed);
      Console.ReadLine();
    }
  }
  class Test
  {
    static void Main(string[] args)
    {
      DaThuc dt = new DaThuc();
      dt.Test();

    }
  }
}
```

B. Bài tập tự làm

Bài 1: Giả sử n ≥ 0 và x là số thực. Hãy tính giá trị của biểu thức sau đây.

$$S(n,x) = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots + \frac{x^n}{n!}$$

Gợi ý:

Ta nhận thấy (x ^ n)/n!=(x.x.x…x)/(n(n-1)(n-2)…2.1) . Để tính S, khai báo 2 biến p và s

Duyệt qua các số tự nhiên từ 1->n ,tại mỗi số ta thực hiện tính tích của thương x và số đó với thương x và các số trước nó (p=p*x/i) sau đó thực hiện cộng dồn các kết quả đó lại ta được tổng S

Algorithm 1: O(N^2)

```
double s=1;
for (int i=1;i<=n;i++)
s=s+pow(x,i)/giaithua(i);// giaithua(i) = i!=1.2.3….i
```

Độ phức tạp theo cách này là O(N^2), tuy nhiên chương trình không thể thực hiện được khi n lớn; chẳng hạn n =100 - do phép tính giai thừa của n không thể thực hiện..

Algorithm 2: O(N^2)

```
double s=1,p;
for (int i=1; i<=n;i++)
{
        p=1;
        for (int j=1; j<=i;j++)
        p=p*x/j;
        s=s+p;
}
```

Độ phức tạp theo cách này vẫn là là O(N^2), tuy nhiên chương trình đã không cần tính giai thừa của n.

Algorithm 3: O(N) – độ phức tạp tuyến tính

```
double s=1,p=1;
for (int i=1;i<=n;i++)
{
        p=p*x/i;
        s=s+p;
}
```

Bài 2: Cho dãy n số nguyên a0,a1,...,an-1. Hãy chuyển k phần tử đầu tiên của dãy về cuối dãy.

<u>Gợi ý</u>:Sử dụng một mảng tg chứa k phần tử và mảng tmp chứa n-k phần tử. Duyệt qua các phần tử a[i] của mảng (i: 0->n):

+Nếu i < k tg[i]=a[i] (chuyển k phần tử đầu sang tg

+Ngược lai,tmp[j++] =a[i] (khởi tạo j=0) chuyển các phần tử còn lại sang tmp

Sau đó thực hiện ghép 2 mảng tmp và tg với nhau

Bài 3: Cho dãy n số nguyên {ai, ở đây giả sử i=1..n} Dãy con liên tiếp là dãy mà thành phần của nó là các thành phần liên tiếp nhau trong {a}, ta gọi tổng của dãy con là tổng tất cả các thành phần của nó. Tìm tổng lớn nhất trong tất cả các tổng của các dãy con của {a}. Ví dụ nếu n = 7;

4 –5 <u>6 –4 2 3</u> -7

Thì kết quả tổng là 7.

Bài 4. Giả sử n ≥1 và x là số thực. Hãy viết hàm tính giá trị của biểu thức sau đây (với độ phức tạp tuyến tính):

$$S(n,x) = \frac{x}{1} - \frac{x^2}{1+\frac{1}{2}} + \frac{x^3}{1+\frac{1}{2}+\frac{1}{3}} - \ldots + (-1)^{n-1}\frac{x^n}{1+\frac{1}{2}+\ldots+\frac{1}{n}}$$

<u>Gợi ý</u>:Xây dựng một hàm tính biểu thức mẫu với đối số là một số tự nhiên bất kỳ,giả sử ta có hàm Mau(int n) trả giá trị 1+1/2+…+1/n

Tại hàm Main ta duyệt qua các số tự nhiên i từ 1->n.Tại mỗi số i ta tính tích của (-1)mũ i với x mũ i và chia cho hàm Mau (với đối số là số i đó), đồng thời tiến hành cộng dồn các kết quả đó lại được S

BÀI 10. THỰC HÀNH THUẬT TOÁN CHIA ĐỂ TRỊ (1)

Mục tiêu

- *Viết được giải thuật theo tư tưởng chia để trị để giải quyết bài toán theo yêu cầu*
- *Cài đặt được giải thuật theo tư tưởng chia để trị để giải quyết bài toán theo yêu cầu*

A. Bài tập mẫu

Bài 1. Cho mảng gồm n phần tử **n phần tử a1, a2,…., an** đã được sắp xếp tăng dần và một phần tử x. Tìm xem x có trong mảng hay không?

Nếu có x trong danh sách thì trả ra kết quả là vị trí tìm thấy,

nếu x không có trong danh sách thì trả ra kết quả là -1

Dùng thuật toán tìm kiếm nhị phân

Hướng dẫn

```
using System;

namespace Binary_search_code
{
  class Program
  {
    public static int BinarySearch(int[] a, int x, int left, int
right)
    {
      if (left > right) return -1;
      else
      {
        int mid = (left + right) / 2;
        if (x == a[mid]) return mid;
        else if (x < a[mid])
          return BinarySearch(a, x, left, mid - 1);
        else
          return BinarySearch(a, x, mid + 1, right);
      }
    }
    static void Main(string[] args)
    {
      Console.WriteLine("Demo BinarySearch\n");
      int[] a = { 1, 4, 5, 7, 9, 10, 23, 50, 67, 88 };
      int n = a.Length;
      int x = 10;
      int index = BinarySearch(a, x, 0, n - 1);
```

```
      if (index == -1)
        Console.WriteLine("Ko có phân tu {0} trong danh sach", x);
      else
        Console.WriteLine("Phân tu {0} nam tai vi tri {1} trong
danh sach", x, index);
      Console.ReadKey();
    }
  }
}
```

Kết quả chương trình chạy

```
C:\Users\diep8\source\repos\Hello2021\Hello2021\bin\Debug\Hello2021.exe
Demo BinarySearch

Phân tu 10 nam tai vi tri 5 trong danh sach
```

B. Bài tập tự làm

Bài 1: Nhập vào danh sách n số ngẫu nhiên. Số x cần tìm

Cài đặt thuật toán BinarySearch và thuật toán tìm kiếm tuần tự (Sequence Search) để kết quả trả về vt, nếu a[vt]=x; vt=-1 nếu ko có phần tử nào của dãy bằng x

Ghi lại và so sánh thời gian chạy của 2 thuật toán khi n đủ lớn (n=100000)

Bài 2: Nhập vào danh sách n số. Cài đặt chương trình tìm số lớn nhấn Max và số nhỏ nhất Min của dãy.

Ghi lại và so sánh thời gian chạy của thuật toán dung chia để chị và 1 thuật toán khác khi n đủ lớn (n=100000)

Bài 3: Cho dãy n số nguyên {ai, ở đây giả sử i=1..n} Dãy con liên tiếp là dãy mà thành phần của nó là các thành phần liên tiếp nhau trong {a}, ta gọi tổng của dãy con là tổng tất cả các thành phần của nó. Tìm tổng lớn nhất trong tất cả các tổng của các dãy con của {a}.

Ví dụ

nếu n = 7; 4 –5 6 –4 2 3 -7 Thì kết quả tổng là 7.

Gợi ý:

Áp dụng chiến lược thiết kế "chia để trị":

(1)Chia đôi mảng thành 2 nửa

(2)Trả về 3 giá trị lớn nhất sau:

a. Tìm tổng lớn nhất của các dãy con của dãy nửa bên trái (gọi đệ quy)

b. Tìm tổng lớn nhất của các dãy con của dãy nửa bên trái (gọi đệ quy)

c. Tìm tổng lớn nhất của các dãy con của dãy xung quanh điểm giữa

BÀI 11. THỰC HÀNH THUẬT TOÁN CHIA ĐỂ TRỊ (2)

Mục tiêu

- *Vận dụng được tư tưởng chia để trị để tìm được cách giải quyết bài toán theo yêu cầu*
- *Cài đặt được các bài toán theo chiến lược chia để trị*

A. Bài tập mẫu

Bài 1. Bài toán sắp xếp dãy số a gồm n phần tử a1, a2,…., an theo thứ tự tăng dần bằng thuật toán Quick Sort

Hướng dẫn

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApplication1
{
  public class TTQuickSort
  {
    //void QuickSort(ref int[] x)
    //{
    //  qs(x, 0, x.Length - 1);
    //}

    void qs(int[] x, int left, int right)
    {
      int i, j;
      int pivot, temp;

      i = left;
      j = right;
      pivot = x[(left + right) / 2];

      do
      {
        while ((x[i] < pivot) && (i < right)) i++;
        while ((pivot < x[j]) && (j > left)) j--;

        if (i <= j)
        {
          temp = x[i];
```

```
          x[i] = x[j];
          x[j] = temp;
          i++; j--;
        }
      } while (i <= j);
      if (left < j) qs(x, left, j);
      if (i < right) qs(x, i, right);
    }
    // Hàm sinh dãy ngẫu nhiên
    void DisplayElements(ref int[] xArray, char status, string
sortname)
    {
      if (status == 'a')
        Console.WriteLine("After sorting using algorithm: " +
sortname);
      else
        Console.WriteLine("Before sorting");
      for (int i = 0; i <= xArray.Length - 1; i++)
      {
        if ((i != 0) && (i % 10 == 0))
          Console.Write("\n");
        Console.Write(xArray[i] + " ");
      }
      Console.ReadLine();
    }
    void MixDataUp(ref int[] x, Random rdn)
    {
      for (int i = 0; i <= x.Length - 1; i++)
      {
        x[i] = (int)(rdn.NextDouble() * x.Length);
      }
    }
    public void Test()
    {
      Console.WriteLine("Sorting Algorithms Demo Code\n\n");

      const int nItems = 20;
      Random rdn = new Random(nItems);
      int[] xdata = new int[nItems];

      MixDataUp(ref xdata, rdn);
      DisplayElements(ref xdata, 'b', "");
      //QuickSort(ref xdata);
      qs(xdata, 0, xdata.Length - 1);

      DisplayElements(ref xdata, 'a', "QuickSort");
      Console.WriteLine("\n\n");
```

```
        Console.ReadLine();
      }
    }

    class Test
    {
      static void Main(string[] args)
      {
        TTQuickSort a = new TTQuickSort();
        a.Test();
        Console.ReadKey();
      }
    }
}
```

Kết quả chương trình chạy

```
C:\Users\diep8\source\repos\Hello2021\Hello2021\bin\Debug\Hello2021.exe
Sorting Algorithms Demo Code


Before sorting
3 13 14 16 4 0 17 9 9 12
0 1 7 17 19 10 5 7 4 1
After sorting using algorithm: QuickSort
0 0 1 1 3 4 4 5 7 7
9 9 10 12 13 14 16 17 17 19
```

B. Bài tập tự làm

Bài 1. Bài toán sắp xếp dãy số a gồm n phần tử a1, a2,…., an theo thứ tự tăng dần bằng thuật toán Merger Sort.

Ghi lại và so sánh thời gian chạy của 2 thuật toán sắp xếp khi n đủ lớn (n=100000)

Bài 2. Cài đặt thuật toán nhân 2 ma trận theo chiến lược chia để trị (Strassen)

BÀI 12. THỰC HÀNH THUẬT TOÁN QUY HOẠCH ĐỘNG

Mục tiêu

- *Vận dụng được chiến lượng quy hoạch động để tìm được cách giải quyết bài toán theo yêu cầu*
- *Cài đặt được các bài toán dãy con chung dài nhất và dãy con trọng lượng lớn nhất theo chiến lược quy hoạch động*

A. Bài tập mẫu

Bài 1: Cho 2 xâu ký tự X, Y. Hãy tìm độ dài dãy con chung lớn nhất của 2 xâu ký tự X,Y (dãy con đảm bảo tính thứ tự nhưng không nhất thiết phải đảm bảo tính liên tiếp).

Chương trình mẫu:

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Text;
using System.Threading.Tasks;

namespace DCchungDaiNhat
{

  class DCchungDaiNhat
  {
    //string X, Y;
    int[,] b;
    int Max(int a, int b)
    {//, out int p){
      return a > b ? a : b;
    }

    void LCS1(string X, string Y, out int[,] c, out int[,] b)
    {
      int m = X.Length, n = Y.Length, i, j;
      c = new int[m + 1, n + 1];
      b = new int[m + 1, n + 1];
      for (i = 0; i <= m; i++) c[i, 0] = 0;
      for (j = 0; i <= n; i++) c[0, j] = 0;

      for (i = 1; i <= m; i++)
      {
        for (j = 1; j <= n; j++)
        {
```

```
          if (X[i - 1] == Y[j - 1])
          {
            c[i, j] = c[i - 1, j - 1] + 1;
            b[i, j] = 0;
          }

          else
          {
            c[i, j] = Max(c[i, j - 1], c[i - 1, j]);
            if (c[i, j] == c[i, j - 1])
              b[i, j] = 1;
            else
              b[i, j] = -1;
          }
          Console.Write(c[i, j] + "," + b[i, j] + "\t");
        }
        Console.WriteLine();
      }

      Console.WriteLine("Day con chung dai nhat cua\n{0}\n{1}\n
Voi do dai la {2}", X, Y, c[m, n]);
    }
    void Show_LCS(string X, string Y, int[,] b)
    {
      int i, j; i = X.Length; j = Y.Length;
      while (!(i == 0 || j == 0))
      {
        if (b[i, j] == 0)
        {
          Console.Write(X[i - 1]);
          i--; j--;
        }

        else if (b[i, j] == 1)
        {
          j--;
        }
        else i--;

      }
    }
    public void Test()
    {
      string X = "AGGTAB", Y = "GXTXAYB";
      int m = X.Length, n = Y.Length;
      Console.WriteLine("Day con chung dai nhat cua\n{0} va
{1}\n", X, Y);
```

```
            int[,] c;
            LCS1(X, Y, out c, out b);
            Show_LCS(X, Y, b);
            Console.ReadKey();
        }
    }

    class test
    {
        static void Main(string[] args)
        {
            DCchungDaiNhat t = new DCchungDaiNhat();
            t.Test();
            Console.ReadKey();
        }
    }
}
```

Kết quả chương trình chạy

C:\Users\diep8\source\repos\Hello2021\Hello2021\bin\Debug\Hello2021.exe

```
Day con chung dai nhat cua
AGGTAB va   GXTXAYB

0,1     0,1     0,1     0,1     1,0     1,1     1,1
1,0     1,1     1,1     1,1     1,1     1,1     1,1
1,0     1,1     1,1     1,1     1,1     1,1     1,1
1,-1    1,1     2,0     2,1     2,1     2,1     2,1
1,-1    1,1     2,-1    2,1     3,0     3,1     3,1
1,-1    1,1     2,-1    2,1     3,-1    3,1     4,0
Day con chung dai nhat cua
AGGTAB
GXTXAYB
 Voi do dai la 4
BATG
```

```
Day con chung dai nhat cua
BEAUTI
DEVINCI
2
0,1     0,1     0,1     0,1     0,1     0,1     0,1
0,1     1,0     1,1     1,1     1,1     1,1     1,1
0,1     1,-1    1,1     1,1     1,1     1,1     1,1
0,1     1,-1    1,1     1,1     1,1     1,1     1,1
0,1     1,-1    1,1     1,1     1,1     1,1     1,1
0,1     1,-1    1,1     2,0     2,1     2,1     2,0
Day con chung dai nhat cua
BEAUTI
DEVINCI
2 Gom:

IE
```

B. Bài tập tự làm

Bài 1: Cài đặt tìm dãy con chung lớn nhất của 2 xâu ký tự X,Y dùng 2 thuật toán. Ghi lại và so sánh thời gian chạy của 2 thuật toán khi dãy đủ lớn

Gợi ý: Tìm dãy con chung bằng giải thuật chia để trị

```
int LCS (string X, string Y, int i, int j)
{
  if (i == 0 || j == 0) return 0;
  else
    if (X[i - 1] == Y[j - 1])
  {
    //b[i, j] = 0;
    return LCS_Chiatri(X, Y, i - 1, j - 1) + 1;

  }
  else
  {
    int a = Max(LCS_Chiatri(X, Y, i, j - 1), LCS_Chiatri(X, Y, i - 1, j));
    //if (a == LCS(X, Y, i, j - 1))
    //  b[i,j]=1;
    //else
    //  b[i, j] = -1;
    return a;
  }
}
```

Bài 2. Cho dãy n số nguyên {ai, ở đây giả sử i=1..n} Dãy con liên tiếp là dãy mà thành phần của nó là các thành phần liên tiếp nhau trong {a}, ta gọi tổng của dãy con là tổng tất cả các thành phần của nó. Tìm tổng lớn nhất trong tất cả các tổng của các dãy con của {a}.

Ví dụ nếu n = 7;4 –5 6 –4 2 3 -7 thì kết quả tổng là 7

Cài đặt bài toán bằng quy hoạch động và 1 thuật toán khác. So sánh 2 thuật toán.

Bài 3. Dùng thuật toán quy hoạch động để tìm trình tự tối ưu nhân dãy các ma trận

Gợi ý!
Nhân 2 ma trận: A m x n và ma trận Bn x p, viết hàm nhân 2 ma trận dựa vào thuật toán:

```
for (int i = 0; i < m; i++)
  for (int j = 0; j < p; j++)
  {
    z[i, j] = 0;
    for (int k = 0; k < n; k++)
      z[i, j] = x[i, k] * y[k, j];
  }
```

Dùng quy hoạch động tìm trình tự nhân dãy ma trận có số phép tính nhân ít nhất

```
namespace multi_matrix
{
  class Program
  {
    // Matrix Ai has dimension p[i-1] x p[i] for i = 1..n
    static int MatrixChainOrder(int[] p, int n)
    {
      /* For simplicity of the program, one extra row and one
      extra column are allocated in m[][]. 0th row and 0th
      column of m[][] are not used */
      int[,] m = new int[n, n];
      int i, j, k, L, q;
      /* m[i,j] = Minimum number of scalar multiplications needed
to compute the matrix A[i]A[i+1]...A[j] = A[i..j] where
dimension of A[i] is p[i-1] x p[i] */
      // cost is zero when multiplying one matrix.
      for (i = 1; i < n; i++)
        m[i, i] = 0;
      // L is chain length.
      for (L = 2; L < n; L++)
      {
        for (i = 1; i < n - L + 1; i++)
        {
          j = i + L - 1;
          m[i, j] = int.MaxValue;
          for (k = i; k <= j - 1; k++)
          {
            // q = cost/scalar multiplications
            q = m[i, k] + m[k + 1, j] + p[i - 1] * p[k] * p[j];
            if (q < m[i, j])
              m[i, j] = q;
```

```
          }
        }
      }
      return m[1, n - 1];
    }
    static void Main(string[] args)
    {
      int[] arr = { 2, 5, 4, 3, 7 }; //số chiều tương ứng của các
ma trận
      int size = arr.Length;
      Console.WriteLine("So phep tinh it nhat la {0} ",
      MatrixChainOrder(arr, size));
      Console.ReadKey();
    }
  }
}
```

BÀI 13. THỰC HÀNH THUẬT TOÁN THAM LAM

Mục tiêu

- *Vận dụng được chiến lược tham để tìm được cách giải quyết bài toán theo yêu cầu*
- *Thiết kế giải thuật tham và cài đặt được các bài toán theo chiến lược tham đã thiết kế.*

A. Bài tập mẫu

Bài 1. Cho n công việc. Công việc thứ I có thời gian bắt đầu là si và thời gian kết thúc là fi. Hãy tìm cách sắp xếp các công việc sao cho số công việc thực hiện là nhiều nhất mà không bị xung đột.

Chương trình mẫu:

```
using System;
namespace Activity___selection_Problem
{
  class Program
  {
    // Prints a maximum set of activities that can be done by a
single
    // person, one at a time.
    // n --> Total number of activities
    // s[] --> An array that contains start time of all activities
    // f[] --> An array that contains finish time of all
activities
    static void printMaxActivities(int[] s, int[] f, int n)
    {
      int i, j;
      Console.WriteLine("Following activities are selected \n");
      // The first activity always gets selected
      i = 0;
      Console.Write("{0} ", i);
      // Consider rest of the activities
      for (j = 1; j < n; j++)
      {
        // If this activity has start time greater than or
        // equal to the finish time of previously selected
        // activity, then select it
        if (s[j] >= f[i])
        {
          Console.Write("{0} ", j);
          i = j;
```

```
            }
        }
    }
    static void Main(string[] args)
    {
      int[] s = { 1, 3, 0, 5, 8, 5 };
      int[] f = { 2, 4, 6, 7, 9, 9 };
      int n = s.Length;
      printMaxActivities(s, f, n);
      Console.ReadKey();
    }
  }
}
```

Kết quả chương trình chạy

```
Following activities are selected

0 1 3 4
```

Bài 2. Bài toán cái tưi KnapSack

Có n vật, mỗi vật i, i=1,…, n được đặc trưng bởi trọng lượng wi và giá trị vi. Có một chiếc túi sách có khả năng mang m kg. Hãy chọn vật xếp vào ba lô sao cho ba lô thu được có giá trị nhất.
Gợi ý! Thiết kế Thuật toán tham lam cho bài toán chọ vật có giá trị giảm dần (theo đơn giá)
Input: W = (w1, w2,…, wn)// Trọng lượng
V = (v1, v2,…,vn)// Giá trị
M = Sức chứa của ba lô.
Output
C[1..n];// Đánh dấu các vật được chọn
Vmax: Giá trị lớn nhất của ba lô

Chương trình mẫu

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Diagnostics;
```

```
namespace ThamLam_Tui
{
  class ThamLam
  {
    static void Nhap(out double[] v, out double[] w, out int n)
    {
      Console.Write("Nhap so luong mat hang co trong quan n= ");
      n = int.Parse(Console.ReadLine());
      v = new double[n];
      w = new double[n];

      Random r = new Random();
      for (int i = 0; i < n; i++)
        v[i] = r.Next(1, 20);
      Random rs = new Random();
      for (int i = 0; i < n; i++)
        w[i] = rs.Next(0, 20);
    }
    static void SapXep(double[] v, double[] w, double[] c)
    {
      double tg = 0;
      for (int i = 0; i < c.Length; i++)
        for (int j = c.Length - 1; j > i; j--)
        {
          if (c[i] < c[j])
          {
            tg = c[i]; c[i] = c[j]; c[j] = tg;
            tg = v[i]; v[i] = v[j]; v[j] = tg;
            tg = w[i]; w[i] = w[j]; w[j] = tg;
          }
        }
    }
    static void Chon(double[] v, double[] w, int m)
    {
      //Console.Write("\n\nNhap khoi luong toi da ma cai tui ten
cuop dung duoc M = ");
      //m = int.Parse(Console.ReadLine());
      int z = 0;   //Số đồ vật
      double WW = 0;
      double VV = 0;
      Console.WriteLine("\nHang lay duoc la: ");
      Console.WriteLine("-----------------------------------------
---\n");
      for (int i = 0; i < v.Length; i++)
      {
        if (WW + w[i] <= m)
        {
```

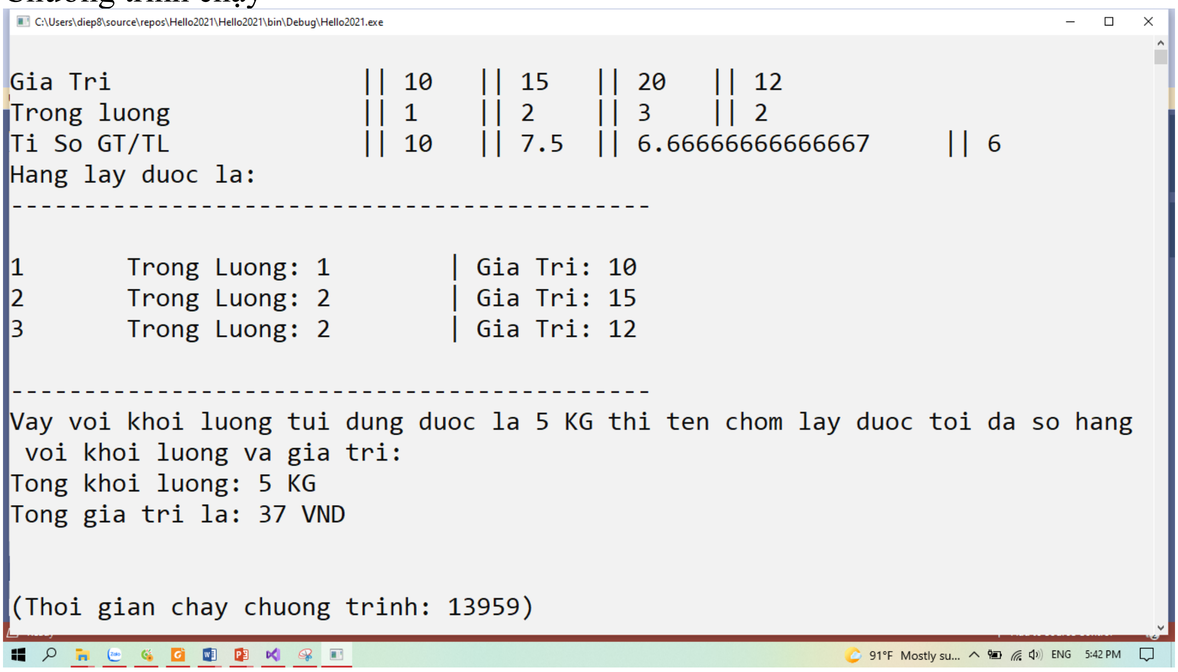
```
          z++;
          Console.WriteLine(z+"\tTrong Luong: " + w[i] + "\t
|\tGia Tri: " + v[i]);
          WW += w[i];
          VV += v[i];
        }
      }
      if (z > 0)
      {
        Console.Write("\n");
        Console.WriteLine("---------------------------------------
-----");
        Console.WriteLine("Vay voi khoi luong tui dung duoc la {0}
KG thi ten chom lay duoc toi da so hang voi khoi luong va gia
tri:", m);
        Console.Write("Tong khoi luong: {0} KG", WW);
        Console.Write("\nTong gia tri la: {0} VNĐ", VV);
      }
      else
      {
        Console.Write("\nTui khong chua duoc vat nao !!!");
      }
    }
    static void HienThi(double[] v, double[] w, double[] c)
    {
      Console.Write("\nGia Tri\t\t");
      for (int i = 0; i < v.Length; i++)
      {
        Console.Write("\t|| " + v[i]);
      }
      Console.Write("\nTrong luong\t");
      for (int i = 0; i < w.Length; i++)
      {
        Console.Write("\t|| " + w[i]);
      }
      Console.Write("\nTi So GT/TL\t");
      for (int i = 0; i < c.Length; i++)
      {
        Console.Write("\t|| " + c[i]);
      }
    }
    static void Main(string[] args)
    {

      double[] v= {12,10,20,15 };
      double[] w= { 2,1,3,2};
      int n = v.Length; int m = 5;
```

```
        double[] c=new double[n];

        for (int i = 0; i < n; i++)
          c[i] = v[i] / w[i];
        //Nhap(out a, out b, out n);
        SapXep(v, w, c);
        HienThi(v, w, c);
        Stopwatch r = new Stopwatch();
        r.Start();
        Chon(v, w, m);
        Console.Write("\n\n\n(Thoi gian chay chuong trinh: {0})",
r.ElapsedTicks);
        Console.ReadKey();
      }
    }
}
```

Chương trình chạy



B. Bài tập tự làm

Bài 1. Cài đặt bài toán rút tiền ở cây ATM

Bài 2. Yêu cầu: Hãy cải tiến chương trình trên bằng cách biểu diễn dữ liệu công việc

```
    class Meeting
    {
        public double Start, End, Duration;
        public string Name;
    }
```

-Nhập dữ liệu cho n công việc
- Sắp xếp các công việc theo thời gian kết thúc tăng dần
- Cài đặt thuật toán tham lam để chọn được nhiều công việc nhất mà không xung đột
- Hiển thị các công việc được chọn gồm đầy đủ thông tin id, start và finish.

BÀI 14. KIỂM TRA THỰC HÀNH

Bài 1 Cho định nghĩa số Fibonaci F(n):

- F(0)=0
- F(1)=1
- F(n)=F(n-2)+F(n-1) với n>1

Hãy viết chương trình sử dụng chia để trị để tính số Fibonaci thứ n (n được lưu trữ ở tệp input.txt – tệp này chứa duy nhất 1 giá trị n). Kết quả tính F(n) được ghi vào tệp output.txt

Bài 2 Sử dụng chiến lược thiết kế chia để trị để viết chương trình tình tổng sau:

$$S = \frac{1}{\sqrt{2}} + \frac{1}{\sqrt{4}} + \cdots .. + \frac{1}{\sqrt{n}}$$

Biết rằng n là một số chẵn và n>=2 (n được lưu trữ ở tệp input.txt – tệp này chứa duy nhất 1 giá trị n). Kết quả của S sau khi tính sẽ được ghi vào tệp output.txt

Bài 3 Cho định nghĩa về tổ hợp C(n,k) như sau:

	C(n-1,k-1) + C(n-1,k)	if o<k<n
C(n,k) =	1	if k =0 or k = n
	0	otherwise

Hãy viết chương trình tính C(n,k) (n,k được lưu trữ ở tệp input.txt – giá trị n, k được ghi trên 2 dòng trên tệp). Kết quả tính C(n,k) được ghi vào tệp output.txt

Bài 4 Nhập vào 1 danh sách n số tang dần. Sử dụng chiến lược thiết kế chia để trị để viết chương trình BinarySearch.

Bài 5 Cho 2 xâu ký tự X, Y. Hãy tìm độ dài dãy con chung lớn nhất của 2 xâu ký tự X,Y (dãy con đảm bảo tính thứ tự nhưng không nhất thiết phải đảm bảo tính liên tiếp).

Yêu cầu:

- Áp dụng chiến lược thiết kế quy hoạch động
- In ra màn hình mảng kết quả của các bước
- Hiển thị ra màn hình xâu con chung dài nhất và độ dài của nó.

Bài 6 The maximum-subarray problem

- Cho dãy n số nguyên {ai, ở đây giả sử i=1..n} Dãy con liên tiếp là dãy mà thành phần của nó là các thành phần liên tiếp nhau trong {a}, ta gọi tổng của dãy con là tổng tất cả các thành phần của nó. Tìm tổng lớn nhất trong tất cả các tổng của các dãy con của {a}. Ví dụ nếu n = 7;
- 4 –5 <u>6 –4 2 3</u> -7
- Thì kết quả tổng là 7.

- Áp dụng thuật toán quy hoạch động viết chương trình in ra dãy con có tổng lớn nhất và giá trị tổng đó.

Bài 7 Cài đặt bài toán rút tiền ở cây ATM

Trong máy rút tiền tự động ATM, ngân hàng đã chuẩn bị sẵn các loại tiền có mệnh giá 500.000 đồng, 200.000 đồng, 100.000 đồng, 50.000 đồng, 20.000 đồng và 10.000 đồng. Giả sử mỗi loại tiền đều có số lượng không hạn chế. Khi có một khách hàng cần rút một số tiền **T** đồng (tính chẵn đến 10.000 đồng, tức là T chia hết cho 10000). Hãy tìm một phương án trả tiền sao cho trả đủ T đồng và số tờ giấy bạc phải trả là ít nhất.

Bài 8 Cho n công việc. Công việc thứ i có thời gian bắt đầu là si và thời gian kết thúc là fi. Hãy tìm cách sắp xếp các công việc sao cho số công việc thực hiện là nhiều nhất mà không bị xung đột

```
struct activity
    {
      string id;
      int start;
      int finish;

    }
```

- Nhập dữ liệu cho n công việc
- Cài đặt thuật toán tham lam để chọn được nhiều công việc nhất mà không xung đột
- Hiển thị các công việc được chọn gồm đầy đủ thông tin id, start và finish.

TÀI LIỆU THAM KHẢO

Tiếng Việt:

1. Trần Tuấn Minh, Khoa Toán Tin, "Thiết kế và đánh giá thuật toán", Trường Đại học Đà Lạt.
2. Vũ Đình Hóa, Đỗ Trung Kiên, "Thiết kế thuật toán", Đại học sư phạm Hà Nội.
3. Đinh Mạnh Tường, "Cấu trúc dữ liệu và thuật toán", Nhà xuất bản khoa học và kỹ thuật, 2000.
4. Robert Sedgewick, Chủ biên dịch: GS. Hoàng Kiếm, "Cẩm nang thuật toán" , NXB KH-KT, 1996.

Tiếng Anh:

5. Robert Lafore, 2001, "Data Structures & Algorithms in Java – SAMS".
6. Algorithms, Robert Sedgewick, Addison-Wesley Publishing, 1983,Garey M.R., Johnson D.S. Computer and intractability.
7. NIKLAUS WIRTH , "Algorithms + data structures = Programs", Prentice-Hall INC,1976