

Thiết kế và đánh giá thuật toán

Bởi:

Khoa CNTT ĐHSP KT Hưng Yên

Thiết kế và đánh giá thuật toán

Bởi:

Khoa CNTT ĐHSP KT Hưng Yên

Phiên bản trực tuyến:

< <http://voer.edu.vn/content/col10114/1.1/> >

Tài liệu này và sự biên tập nội dung có bản quyền thuộc về Khoa CNTT ĐHSP KT Hưng Yên. Tài liệu này tuân thủ giấy phép Creative Commons Attribution 3.0 (<http://creativecommons.org/licenses/by/3.0/>).

Tài liệu được hiệu đính bởi: July 23, 2010

Ngày tạo PDF: July 24, 2010

Để biết thông tin về đóng góp cho các module có trong tài liệu này, xem tr. 88.

1 Lời nói đầu	
1.1 Lời nói đầu	1
2 Bài 1: Thuật toán và độ phức tạp	
2.1 Giới thiệu môn học, phương pháp học	3
2.2 Khái niệm thuật toán:	3
2.3 Các vấn đề liên quan đến thuật toán	4
2.4 Biểu diễn thuật toán bằng phương pháp sơ đồ	5
3 Bài 2:Phân tích thuật toán	
3.1 Phân tích thuật toán	9
3.2 Độ phức tạp của thuật toán	10
3.3 Một số phương pháp thiết kế	15
4 Bài 3: Cơ bản về thuật toán chia để trị	
4.1 Cơ bản về thuật toán chia để trị	17
4.2 Sơ đồ chung của thuật toán	17
4.3 Tìm kiếm nhị phân	18
4.4 Bài toán Min_Max	20
5 Bài 4: Các bài toán sử dụng thuật toán chia để trị (Divid and Conquer)	
5.1 Thuật toán nhân 2 ma trận	23
5.2 Thuật toán sắp xếp	25
5.3 Bài toán hoán đổi	32
6 Bài 5: Cơ bản về thuật toán Quay Lui (Back Tracking)	
6.1 Sơ đồ chung của thuật toán quay lui	35
6.2 Bài toán ngựa đi tuần	37
6.3 Bài toán 8 quân hậu	40
6.4 Bài toán tìm kiếm đường đi trên đồ thị	42
6.5 Một số bài toán khác	43
7 Bài 6: Cơ bản về thuật toán nhánh cận và các bài toán	
7.1 Sơ đồ chung của thuật toán	47
7.2 Bài toán người du lịch	48
7.3 Bài toán cái túi xách	51
8 Bài 7: Cơ bản về thuật toán Tham Lam	
8.1 Thuật toán Tham Lam	57
8.2 Bài toán người du lịch	58
8.3 Thuật toán Dijkstra – Tìm đường đi ngắn nhất trong đồ thị có trọng số	62
9 Bài 8: Cơ bản về thuật toán Quy hoạch động	
9.1 Sơ đồ chung của thuật toán	67
9.2 Bài toán thực hiện dãy phép nhân ma trận	68
10 Bài 9: Các bài toán sử dụng thuật toán Quy hoạch động	
10.1 Tập độc lớn nhất trên cây	73
10.2 Bài toán dãy con lớn nhất	75
10.3 Bài toán dãy con chung dài nhất	76

11 Bài 10: Bài tập và tổng kết	
11.1 Bài tập tổng kết	79
12 Tài liệu tham khảo	
12.1 Tài liệu tham khảo	87
Attributions	88

Chương 1

Lời nói đầu

1.1 Lời nói đầu¹

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

¹This content is available online at <<http://voer.edu.vn/content/m12773/1.1/>>.

Chương 2

Bài 1: Thuật toán và độ phức tạp

2.1 Giới thiệu môn học, phương pháp học¹

Đây là module cung cấp cho người học những nguyên lý cơ bản về lập trình, các cấu trúc điều khiển, các kiểu dữ liệu, mô hình hướng chức năng, cách thức xây dựng chương trình con và một số bài toán trong khoa học kỹ thuật.

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:

- Giải thích được các nguyên lý cơ bản về lập trình máy
- Giải thích và mô tả được cú pháp, nguyên tắc hoạt động và cách sử dụng các cấu trúc điều khiển, chương trình con. Chỉ ra được đặc điểm và cách sử dụng của các kiểu dữ liệu
- Mô tả được thuật toán và biểu diễn thuật toán dưới dạng lưu đồ
- Phân tích, thiết kế và cài đặt được bài toán theo mô hình hướng chức năng
- Vận dụng được các kiến thức đã học để cài đặt các bài toán đơn giản trong khoa học kỹ thuật
- Hình thành thái độ học tập nghiêm túc, kỹ năng làm việc độc lập và kỹ năng làm việc độc lập và làm việc theo nhóm.

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 và vận dụng nó một cách linh hoạt vào các trường hợp cụ thể, phải làm nhiều bài tập...

2.2 Khái niệm thuật toán:²

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 Ả-rập 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:

2.2.1 Thuật toán Euclid .

Input: **m, n** nguyên dương

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

¹This content is available online at <<http://voer.edu.vn/content/m12765/1.2/>>.

²This content is available online at <<http://voer.edu.vn/content/m12769/1.1/>>.

2.2.2 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 ($r \neq 0$), 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 nguyên, 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 điển Oxford Dictionary định nghĩa, Algorithm: set of well - defined rules for solving a problem in a finite number of steps.)

2.3 Các vấn đề liên quan đến thuật toán³

2.3.1 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-conquer), 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.

2.3.2 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 m 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.

2.3.3 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.

³This content is available online at <<http://voer.edu.vn/content/m12666/1.1/>>.

2.3.4 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:

Thuật toán đơn giản, dễ hiểu, dễ cài đặt (dễ viết chương trình)

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 thuật 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.

2.4 Biểu diễn thuật toán bằng phương pháp sơ đồ

2.4.1 Phương pháp liệt kê từng bước⁴

Có nhiều phương pháp biểu diễn thuật toán. Có thể biểu diễn thuật toán bằng danh sách các bước, các bước được diễn đạt bằng ngôn ngữ thông thường và các ký hiệu toán học. Có thể biểu diễn thuật toán bằng sơ đồ khối. Tuy nhiên, để đảm bảo tính xác định của thuật toán như đã trình bày trên, thuật toán cần được viết trên các ngôn ngữ lập trình. Một chương trình là sự biểu diễn của một thuật toán trong ngôn ngữ lập trình đã chọn. Thông thường ta dùng ngôn ngữ lập trình Pascal, một ngôn ngữ thường được chọn để trình bày các thuật toán trong sách báo.

Ngôn ngữ thuật toán là ngôn ngữ dùng để miêu tả thuật toán. Thông thường ngôn ngữ thuật toán bao gồm ba loại:

- + Ngôn ngữ liệt kê từng bước;
- + Sơ đồ khối;
- + Ngôn ngữ lập trình;

Ngôn ngữ liệt kê từng bước nội dung như sau:

Thuật toán: Tên thuật toán và chức năng.

Vào: Dữ liệu vào với tên kiểu.

Ra: Các dữ liệu ra với tên kiểu.

Biến phụ (nếu có) gồm tên kiểu.

Hành động là các thao tác với các lệnh có nhân là các số tự nhiên.

Ví dụ. Để giải phương trình bậc hai $ax^2 + bx + c = 0$, ta có thể mô tả thuật toán bằng ngôn ngữ liệt kê như sau:

Bước 1: Xác định các hệ số a,b,c.

Bước 2: Kiểm tra xem các hệ số a,b,c có khác 0 hay không?

Nếu a=0 quay lại thực hiện bước 1.

Bước 3: Tính biểu thức $\Delta = b^2 - 4*a*c$.

Bước 4: Nếu $\Delta < 0$ thông báo phương trình vô nghiệm và chuyển sang bước 8.

⁴This content is available online at <<http://voer.edu.vn/content/m12653/1.1/>>.

Bước 5: Nếu $\Delta=0$, tính $x_1=x_2=\frac{-b}{2a}$ và chuyển sang bước 7.

Bước 6: Tính $x_1=\frac{-b-\sqrt{\Delta}}{2a}$, $x_2=\frac{-b+\sqrt{\Delta}}{2a}$ và chuyển sang bước 7.

Bước 7: Thông báo các nghiệm x_1, x_2 .

Bước 8: Kết thúc thuật toán.

2.4.2 Phương pháp sơ đồ⁵

Phương pháp dùng sơ đồ khối mô tả thuật toán là dùng mô tả theo sơ đồ trên mặt phẳng các bước của thuật toán. Sơ đồ khối có ưu điểm là rất trực giác dễ bao quát.

Để mô tả thuật toán bằng sơ đồ khối ta cần dựa vào các nút sau đây:

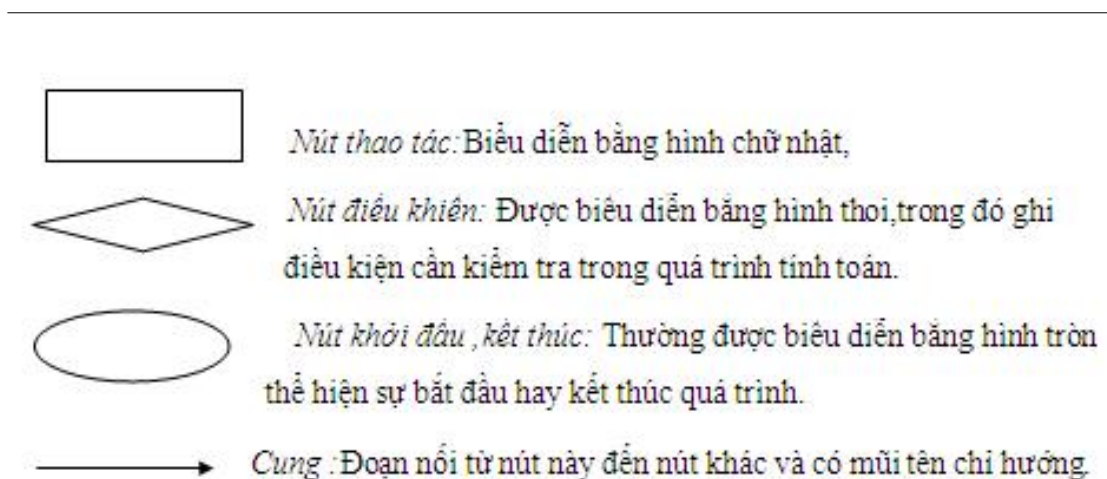


Figure 2.1

Hoạt động của thuật toán theo lưu đồ được bắt đầu từ nút đầu tiên. Sau khi thực hiện các thao tác hoặc kiểm tra điều kiện ở mỗi nút thì bộ xử lý sẽ theo một cung để đến nút khác. Quá trình thực hiện thuật toán dừng khi gặp nút kết thúc hay nút cuối.

Ví dụ. : Để giải phương trình bậc hai $ax^2+bx+c=0$ ta có thể mô tả thuật toán bằng sơ đồ khối sau:

⁵This content is available online at <<http://voer.edu.vn/content/m12772/1.1/>>.

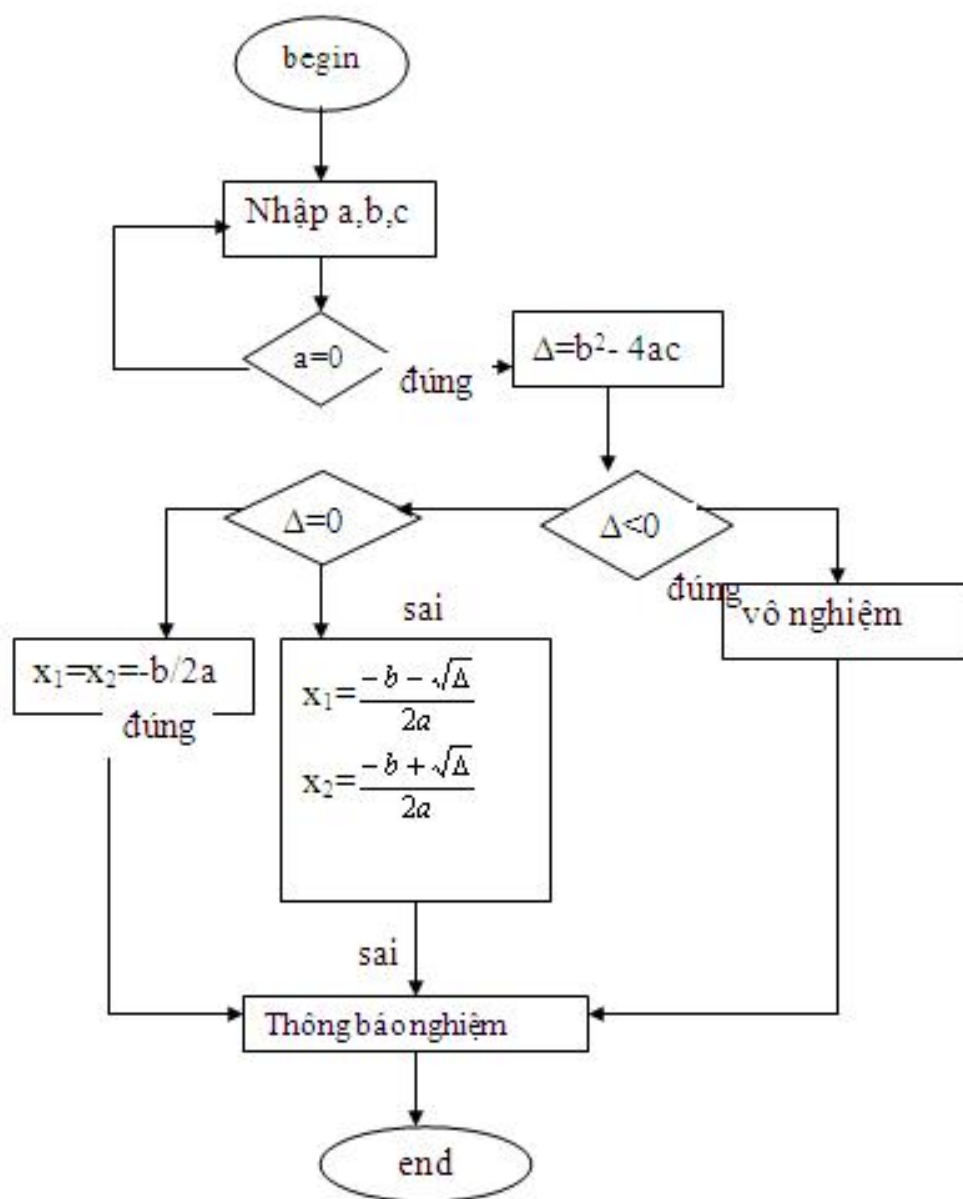


Figure 2.2

Ví dụ biểu diễn bằng lưu đồ thuật toán Euclid:

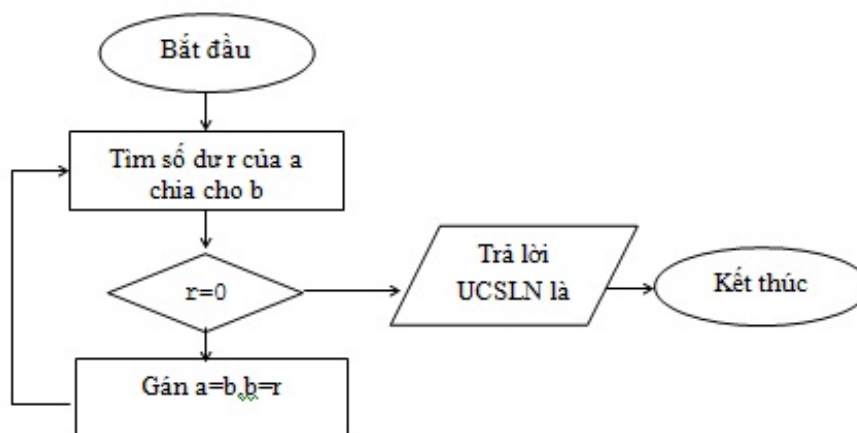


Figure 2.3

2.4.3 Mã giả (pseudocode)⁶

Để diễn đạt một giải thuật có thể sử dụng nhiều loại ngôn ngữ lập trình khác nhau. Thông thường người ta hay sử dụng các ngôn ngữ lập trình cấp cao như Pascal, C, C++, C#, Java . . . Nhưng để sử dụng các ngôn ngữ đó ta gặp phải một số hạn chế sau :

- + Phải luôn tuân thủ các qui luật chặt chẽ về cú pháp của ngôn ngữ đó, khiến cho việc trình bày giải thuật và cấu trúc có thiên hướng nặng nề, gò bó.
- + Phải phụ thuộc vào cấu trúc dữ liệu tiền định của ngôn ngữ, nên có lúc không thể hiện đầy đủ các ý về cấu trúc mà ta muốn biểu đạt.
- + Ngôn ngữ nào được chọn cũng không hẳn đã được mọi người ưa thích và muốn sử dụng.

Để diễn đạt giải thuật một cách tự do hơn, phù hợp với tất cả mọi người sử dụng với một mức độ linh hoạt nhất định, không quá gò bó, không câu nệ về cú pháp và gần gũi với các ngôn ngữ chuẩn để khi cần thiết ta có thể dễ dàng chuyển đổi ta sử dụng ngôn ngữ gần giống với một ngôn ngữ lập trình nào đó gọi là "mã giả"

Ví dụ viết mã giả cho thuật toán giải phương trình bậc 2 như sau:

Read(a); {nhập cho đến khi a=0}

Read(b);

Read(c);

$\Delta = b^2 - 4*a*c;$

If $\Delta < 0$ then Phương trình vô nghiệm

Else if $\Delta = 0$ Phương trình có nghiệm kép
else Phương trình có hai nghiệm phân biệt

⁶This content is available online at <<http://voer.edu.vn/content/m12660/1.1/>>.

Chương 3

Bài 2: Phân tích thuật toán

3.1 Phân tích thuật toán¹

3.1.1 Đặt vấn đề

Khi xây dựng được thuật toán để giải bài toán thì có hàng 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.

3.1.2 Phân tích đánh giá thời gian chạy của thuật toán

- Bước đầu tiên phân tích thời gian chạy của thuật toán là quan tâm 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 trườ 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,

¹This content is available online at <<http://voer.edu.vn/content/m12783/1.1/>>.

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 thứ tự ngược với thứ tự cần sắp xếp

Dãy có thứ tự ngẫu nhiên

3.2 Độ phức tạp của thuật toán

3.2.1 $O(f(x))$ và đánh giá thời gian thực hiện thuật toán.²

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 n_0 sao cho $T(n) \leq c.f(n)$, với mọi $n \geq n_0$.

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

$$3n^2 + 4n + 5 \leq 3n^2 + 4n^2 + 5n^2 = 12n^2, \text{ với mọi } n \geq 1.$$

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 $f_1(n)$ nên tồn tại các hằng số c_0, n_0, c_1, n_1 sao cho $T(n) \leq c_0 f(n)$ với mọi $n \geq n_0$ và $f(n) \leq c_1 f_1(n)$ với mọi $n \geq n_1$. Từ đó ta có $T(n) \leq c_0 c_1 f_1(n)$ với mọi $n \geq \max(n_0, n_1)$.

Khi biểu diễn cấp của thời gian 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 $T(n) = O(f(n))$. Thông thường $f(n)$ là các hàm số sau đây: $f(n)=1$; $f(n)=\log n$; $f(n)=n$; $f(n)=n\log(n)$; $f(n)=n^2$; $n^3 \dots$; $f(n)=2^n$.

- 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 $T(n) \leq cn$ 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.

²This content is available online at <<http://voer.edu.vn/content/m12785/1.1/>>.

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

Figure 3.1

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 , $n \log_2 n$ thường được gọi là các hàm đa thức. Thuật toán với thời gian chạy có cấp hàm đa thức thường chấp nhận được

3.2.2 Các qui tắc để đánh giá thời gian thực hiện thuật toán³

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 $T_1(n) \leq c_1 f_1(n)$, $T_2(n) \leq c_2 f_2(n)$ với mọi $n \geq n_1$, mọi $n \geq n_2$. Đặt $n_0 = \max(n_1, n_2)$.

Khi đó, với mọi $n \geq n_0$ ta có bất đẳng thức sau:

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

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

³This content is available online at <<http://voer.edu.vn/content/m12789/1.1/>>.

có thời gian đánh giá 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 quy 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 S_1, S_2, \dots, S_n là câu lệnh thì

begin S_1, S_2, \dots, S_n **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 S_1, S_2 là các câu lệnh và E là biểu thức logic thì :

If E **then** S_1 **else** S_2

Và

if E **then** S_1

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 S_1, S_2 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 S_1, S_2, \dots, S_n là các câu lệnh, E là biểu thức có kiểu thứ tự đếm được, và v_1, v_2, \dots, v_n là các giá trị có cùng kiểu với E thì :

Case E of

v_1 : S_1 ;

v_2 : S_2 ;

.....

v_n : S_n ;;

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 logic 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 S_1, S_2, \dots, S_n là các câu lệnh, E là biểu thức logic thì

Repeat S_1, S_2, \dots, S_n **until** E

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

Giả sử, thời gian thực hiện khối **begin** S_1, S_2, \dots, S_n **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 E_1, E_2 là biểu thức cùng một kiểu thứ tự đếm được thì

For $i := E_1$ **to** E_2 **do** S là câu lệnh, và

for $i := E_2$ **downto** E_1 **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**.

3.2.3 Đánh giá thủ tục (hoặc hàm) đệ quy.⁴

Ví dụ: Đánh giá thời gian thực hiện của hàm đệ quy sau:

⁴This content is available online at <<http://voer.edu.vn/content/m12795/1.1/>>.

```

(hàm tính n!)
Function fact(n:integer):integer;
Begin
If n<=1 then fact :=1
Else fact:=n*fact(n-1);
End;

```

Trong hàm này cỡ của dữ liệu vào là n . Giả sử thời gian thực hiện hàm là $T(n)$. Với $n=1$ chỉ cần thực hiện lệnh gán $\text{fact}:=1$; do đó $T(1) = O(1)$. Với $n>1$, cần thực hiện lệnh gán $\text{fact}:=n*(\text{fact}(n-1))$. Do đó, thời gian $T(n)$ sẽ là $O(1)$ để thực hiện phép nhân (*) và phép gán(:=) cộng với thời gian $T(n-1)$ để thực hiện lời gọi đệ qui $\text{fact}(n-1)$. Tóm lại ta có quan hệ đệ qui như sau:

```

T(1) = O(1);
T(n) = O(1) + T(n-1);

```

Thay các $O(1)$ bởi các hằng nào đó ta có quan hệ đệ qui như sau:

```

T(1) = C1 ;
T(n) = C2 + T(n-1);

```

Để giải phương trình đệ qui, tìm $T(n)$, chúng ta áp dụng phương pháp thế lặp. Ta có phương trình đệ qui sau:

```

T(m) = C2 + T(m-1); với m >1

```

Thay m lần lượt bởi các 2, 3,..., $n-1, n$ ta được hệ các quan hệ sau:

```

T(2) = C2 + T(1);
T(3) = C2 + T(2);

```

```

.....
T(n-1) = C2 + T(n-2);
T(n) = C2 + (n-1) ;

```

Bằng các phép thế liên tiếp ta nhận được

```

T(n) = (n-1).C2 + T(1)

```

Hay $T(n) = (n-1) C2 + C1$;, trong đó $C1, C2$ là các hằng nào đó .

Do đó $T(n) = O(n)$;

Từ đó ta có phương pháp tổng quát sau đây để đánh giá thời gian thực hiện các thủ tục (hàm) đệ qui. Ta giả thiết rằng các thủ tục (hàm) là đệ qui trực tiếp. Điều đó có nghĩa là các thủ tục (hàm) chỉ chứa các lời gọi đệ qui đến chính nó (không qua một thủ tục hoặc hàm nào khác cả). Giả sử thời gian thực hiện thủ tục (hàm) là $T(n)$, với n là cỡ dữ liệu vào. Khi đó thời gian thực hiện các lời gọi đệ qui thủ tục sẽ là $T(m)$, với $m < n$. Đánh giá thời gian $T(n_0)$ với n_0 là cỡ dữ liệu vào nhỏ nhất có thể được (trong ví dụ trên đó là $T(1)$). Sau đó đánh giá thân của thủ tục theo các qui tắc đã nêu trên ,ta sẽ nhận được quan hệ đệ qui như sau:

```

T(n) = F(T(m1),T(m2),...,T(mk))

```

Trong đó $m_1, m_2, \dots, m_k < n$. Giải phương trình đệ qui này ta sẽ nhận được sự đánh giá của $T(n)$.

Bây giờ ta sử dụng những kiến thức trên để đánh giá hai ví dụ quen thuộc sau đây:

Ví dụ. Xác định độ phức tạp thuật toán của hàm tính dãy số Fibonacci:

```

Function Fibo (n : integer) : integer;
Var i , j , k : integer ;
Begin
i := 1;
j := 0 ;
for k := 1 to n do
Begin
j := i + j ;
i := j - i ;
End;
Fibo := j;
End;

```

Thời gian thực hiện của lệnh trên là $O(n)$.

Một bài toán thường có nhiều cách giải, hay nhiều thuật toán để giải, với mỗi thuật toán khác nhau có thể sẽ có độ phức tạp khác nhau. Đánh giá độ phức tạp thuật toán là một trong những cách phân tích, so sánh và tìm ra trong những thuật toán đó một thuật toán tối ưu.

Ví dụ. Xét bài toán : Tính giá trị đa thức :

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

Xét thuật toán 1 :

Tính giá trị từng hạng tử của đa thức :

1. Với $i = 1$ đến n tính $a_i x_0^i$.

2. Tính Đa thức $P(x)$ có thể viết dưới dạng :

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

Xét Thuật toán 2:

1. $P := a_n$.

2. Với $i=1$ đến n : $P := P x_0 + a_{n-i}$

3. Gán kết quả $P(x_0) = P$.

Chẳng hạn với $n = 3$.

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

1. Tính $P := a_3$

2. $i = 1$: $P = (a_3 x_0 + a_2)$

$i = 2$: $P = (a_3 x_0 + a_2)x + a_1$

$i = 3$: $P = ((a_3 x_0 + a_2)x + a_1)x + a_0$

3. $P(x_0) = ((a_3 x_0 + a_2)x + a_1)x + a_0$

Xét độ phức tạp của hai thuật toán trên :

Thuật toán 1: ở bước 1 ta có :

$i = 1$: ta phải thực hiện 1 phép nhân.

$i = 2$: ta phải thực hiện 2 phép nhân.

.....

$i = n$: ta phải thực hiện n phép nhân.

Như vậy ở bước 1 ta sẽ phải thực hiện $1 + 2 + \dots + n =$

ở bước 2 ta có : ta phải thực hiện n phép cộng.

Vậy ở thuật toán 1 cần $\frac{n(n+1)}{2} + n = \frac{n(n+3)}{2}$

Thuật toán 2: ta phải thực hiện n lần mỗi lần đòi hỏi 2 phép tính (một phép tính cộng và một phép tính nhân). Như vậy thuật toán 2 cần tất cả $2n$ phép tính. Nếu ta coi thời gian thực hiện mỗi phép tính nhân và tính cộng là như nhau và là một đơn vị thời gian thì thời gian thực hiện thuật toán 1 là , còn thời gian thực hiện thuật toán 2 là $2n$.

Như vậy theo phân tích ở trên ta thấy rằng thời gian thực hiện thuật toán 2 ít hơn so với thời gian thực hiện thuật toán một. Thuật toán 2 có độ phức tạp là $O(n)$, độ phức tạp tuyến tính, còn thuật toán 1 thì độ phức tạp là $O(n^2)$ độ phức tạp bậc hai.

Ta nhận thấy rằng việc đánh giá độ phức tạp của một thuật toán nào đó là việc không hề đơn giản nó đòi hỏi phải có phương pháp và cách tiếp cận riêng.

Ví dụ. Phân tích thuật toán Euclide tìm ước số chung lớn nhất của hai số nguyên dương a, b .

Thuật toán Euclide :

Input : a, b là hai số nguyên dương.

Output : ước số chung lớn nhất của hai số a, b .

Function USCLN(a, b)

Begin

$x := a;$

$y := b;$

While $y \neq 0$

begin

$r := x \bmod y$

```

x := y;
y := r;
end;
USCLN := x;
End;

```

Để đánh giá độ phức tạp của thuật toán trên, ta đếm số phép chia thực hiện theo thuật toán.

3.3 Một số phương pháp thiết kế⁵

Trên cơ sở lý thuyết máy Turing, ta chia được các bài toán thành 2 lớp không giao nhau : Lớp giải được bằng thuật toán , và lớp không giải được bằng thuật toán.

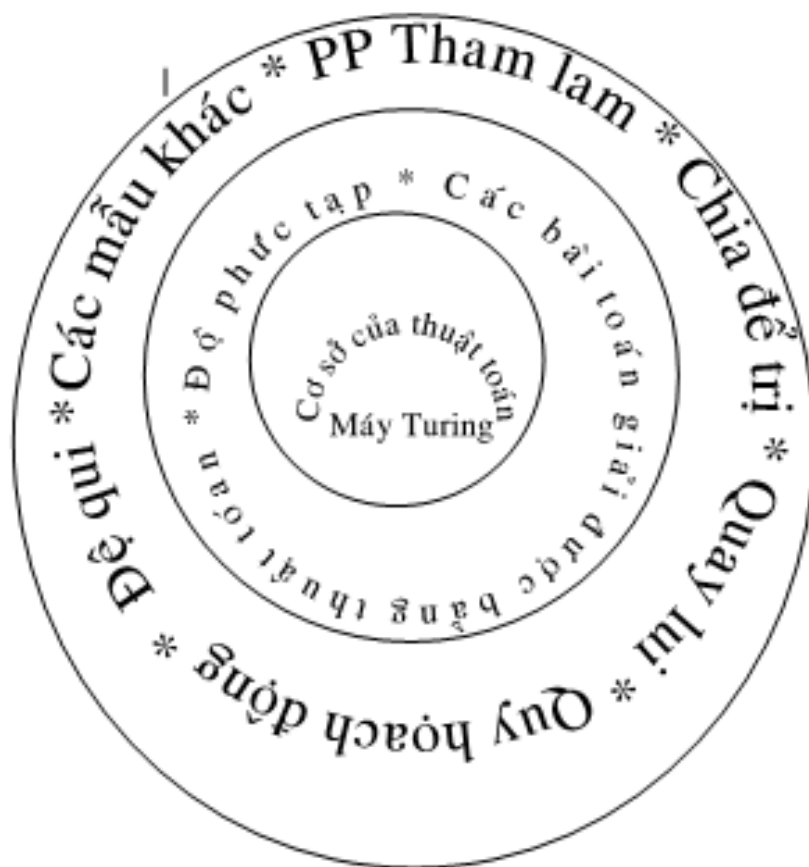


Figure 3.2

⁵This content is available online at <<http://voer.edu.vn/content/m12799/1.1/>>.

Đối với lớp các bài toán giải được bằng thuật toán, dựa vào các đặc trưng của quá trình thiết kế của thuật toán, ta có thể chỉ ra một số các phương pháp thiết kế thuật toán cơ bản sau đây :

3.3.1 Phương pháp chia để trị. (Divide-and-Conquer method).

Ý tưởng là : Chia dữ liệu thành từng miền đủ nhỏ, giải bài toán trên các miền đã chia rồi tổng hợp kết quả lại .

Chẳng hạn như thuật toán Quicksort.

3.3.2 Phương pháp quay lui (BackTracking method).

Ý tưởng là: Tìm kiếm theo ưu tiên. Đối với mỗi bước thuật toán, ưu tiên theo độ rộng hay chiều sâu để tìm kiếm.

Chẳng hạn thuật toán giải bài toán 8 hậu.

3.3.3 Phương pháp tham lam (Greedy Method).

Ý tưởng là : Xác định trật tự xử lý để có lợi nhất, Sắp xếp dữ liệu theo trật tự đó, rồi xử lý dữ liệu theo trật tự đã nêu. Công sức bỏ ra là tìm ra trật tự đó.

Chẳng hạn thuật toán tìm cây bao trùm nhỏ nhất (Shortest spanning Trees).

3.3.4 Phương pháp Quy hoạch động (Dynamic Programming method).

Phương pháp quy hoạch động dựa vào một nguyên lý, gọi là nguyên lý tối ưu của Bellman :

- Nếu lời giải của bài toán là tối ưu thì lời giải của các bài toán con cũng tối ưu. Phương pháp này tổ chức tìm kiếm lời giải theo kiểu từ dưới lên. Xuất phát từ các bài toán con nhỏ và đơn giản nhất, tổ hợp các lời giải của chúng để có lời giải của bài toán con lớn hơn...và cứ như thế cuối cùng được lời giải của bài toán ban đầu.

Chẳng hạn thuật toán “chiếc túi xách” (Knapsack).

Chương 4

Bài 3: Cơ bản về thuật toán chia để trị

4.1 Cơ bản về thuật toán chia để trị¹

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 thủy. 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.

4.2 Sơ đồ chung của thuật toán²

4.2.1 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 vấn đề cốt lõi 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 tối 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)

4.2.2 Sơ đồ tổng quát thuật toán chia để trị

(Viết theo tựa Pascal):

```
Procedure D_and_C(n)
Begin
  If  $n \leq n_0$  Then
    Giải bài toán một cách trực tiếp
  Else
```

¹This content is available online at <<http://voer.edu.vn/content/m12811/1.1/>>.

²This content is available online at <<http://voer.edu.vn/content/m12817/1.1/>>.

- Chia bài toán thành r bài toán con kích thước n/k
- For (Mỗi bài toán trong r bài toán con) Do D_and_C(n)
- 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 ;
  }
}
```

4.3 Tìm kiếm nhị phân³

4.3.1 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,

4.3.2 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:

³This content is available online at <<http://voer.edu.vn/content/m12836/1.1/>>.


```

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) ;

    }

```

Figure 4.1

4.3.3 Đánh giá độ phức tạp thời gian của thuật toán

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

=> Ta có : $T_{\text{tốt}}(n) = O(1)$

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 ($x == a[\text{giữa}]$) 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 $n \geq 2$ và $T(1) = 0$

4.4 Bài toán Min_Max⁴

4.4.1 Bài toán

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

4.4.2 Phân tích thuật toán

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

Figure 4.2

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

Mình họa :

Tìm giá trị Min, Max trong đoạn $a[2..7]$ của mảng $a[1..7]$.

Ký hiệu :

MinMax($a, l, r, \text{Min}, \text{Max}$) cho Min và Max trong đoạn $a[l..r]$

MinMax($a, 2, 7, \text{Min}, \text{Max}$) cho Min = 0 và Max = 15 trong đoạn $a[2..7]$

Mô tả thuật toán:

Input : $a[l..r]$, ($1 \leq r$)

Output: Min = Min ($a[l], \dots, a[r]$),

Max = Max ($a[l], \dots, a[r]$).

MinMax($a, l, r, \text{Min}, \text{Max}$)

if ($l == r$)

{

Min = $a[l]$;

Max = $a[l]$;

}

Else

{

MinMax($a, l, (l+r) / 2, \text{Min1}, \text{Max1}$);

MinMax($a, (l+r) / 2 + 1, r, \text{Min2}, \text{Max2}$);

If ($\text{Min1} < \text{Min2}$)

Min = Min1;

Else

Min = Min2;

If ($\text{Max1} > \text{Max2}$)

Max = Max1;

Else

Max = Max2;

}

⁴This content is available online at <<http://voer.edu.vn/content/m12830/1.1/>>.

4.4.3 Độ phức tạp thuật toán

Gọi $T(n)$ là số phép toán so sánh cần thực hiện. Khi đó ta có:

$$T(n/2) + T(n/2) + 2; n > 2$$

$$1 \quad ; \quad n = 2$$

$$0 \quad ; \quad n = 1$$

$$T(n) = \{ \{$$

Với $n=2^k$, thì:

$$T(n) = 2 + 2T(n/2) = 2 + 2^2 + 2^2 T(n/2^2) = 2^{k-1}T(2) + \sum_{i=1}^{k-1} 2^i$$

$$= \sum_{i=1}^k 2^i - 2^{k-1} = 2^{k+1} - 2^{k-1} - 2 = \frac{3n}{2} - 2$$

Vậy $T(n) \in O(n)$.

Chương 5

Bài 4: Các bài toán sử dụng thuật toán chia để trị (Divid and Conquer)

5.1 Thuật toán nhân 2 ma trận¹

5.1.1 Bài toán :

Cho hai ma trận A, B với kích thước $n \times n$, 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:

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

Figure 5.1

5.1.2 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 $n \times n$) thì thời gian để tính phép nhân ma trận AB là (n^3) .

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 $A \times B = C$ theo các ma trận con đó:

¹This content is available online at <<http://voer.edu.vn/content/m12870/1.1/>>.

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

Figure 5.2

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 thức 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 quy $n/2 \times n/2$ ma trận và $O(n^2)$ phép cộng trừ vô hướng theo công thức truy hồi :

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

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

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

Figure 5.3

- 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 $n \times n$ là $O(n^\omega)$. 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 $n \times n$ với thời gian là $O(n^{\log_3 21})$ 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 trận 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 $\lg 7$. 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 $O(n^{2525813})$. Hãy tưởng tượng rằng, ngay sau đó tháng 1 năm 1980 thời gian tính của phép nhân ma trận là $O(n^{2525813})$. Biểu thức tiệm cận thời gian tính toán tồi nhất của thuật toán nhân ma trận kích thước $n \times n$ được Coppersmith và Winograd phát minh ra năm 1986 là $O(n^{2376})$. 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.

5.2 Thuật toán sắp xếp²

5.2.1 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 $n \log n$.

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.

²This content is available online at <<http://voer.edu.vn/content/m12855/1.1/>>.

5.2.2 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.

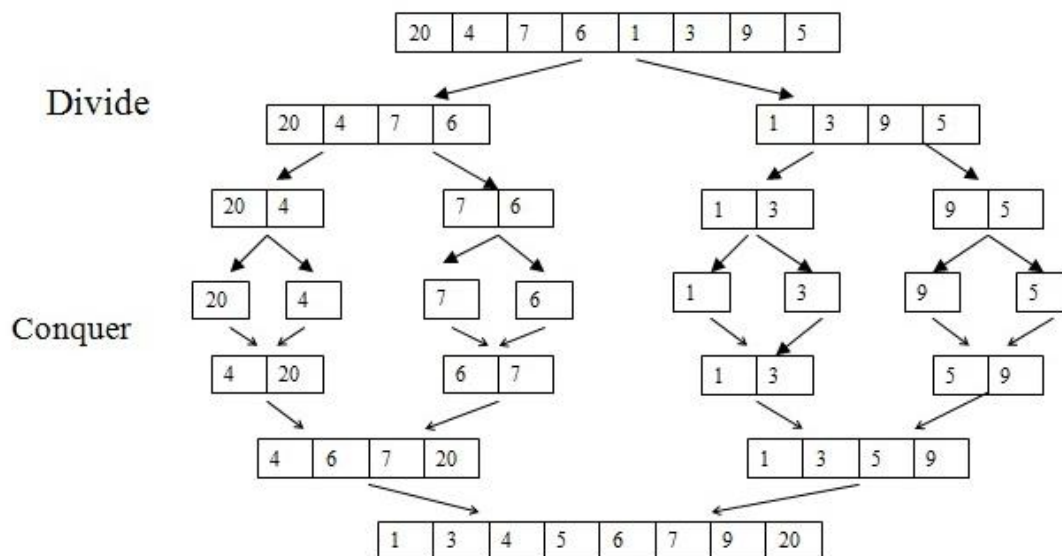


Figure 5.4

Mảng đã được sắp theo thứ tự tăng

Giải thuật sắp xếp này minh họa 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.

* 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.

5.2.3 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 quyết điều đó đơn giản là chúng ta sử dụng 1 phần tử tùy 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 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 $T[l] < p$. Tiếp đến hoán vị $T[k]$ và $T[l]$. Quá trình này tiếp tục cho đến khi $k \geq l$. 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
  }
}
```

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

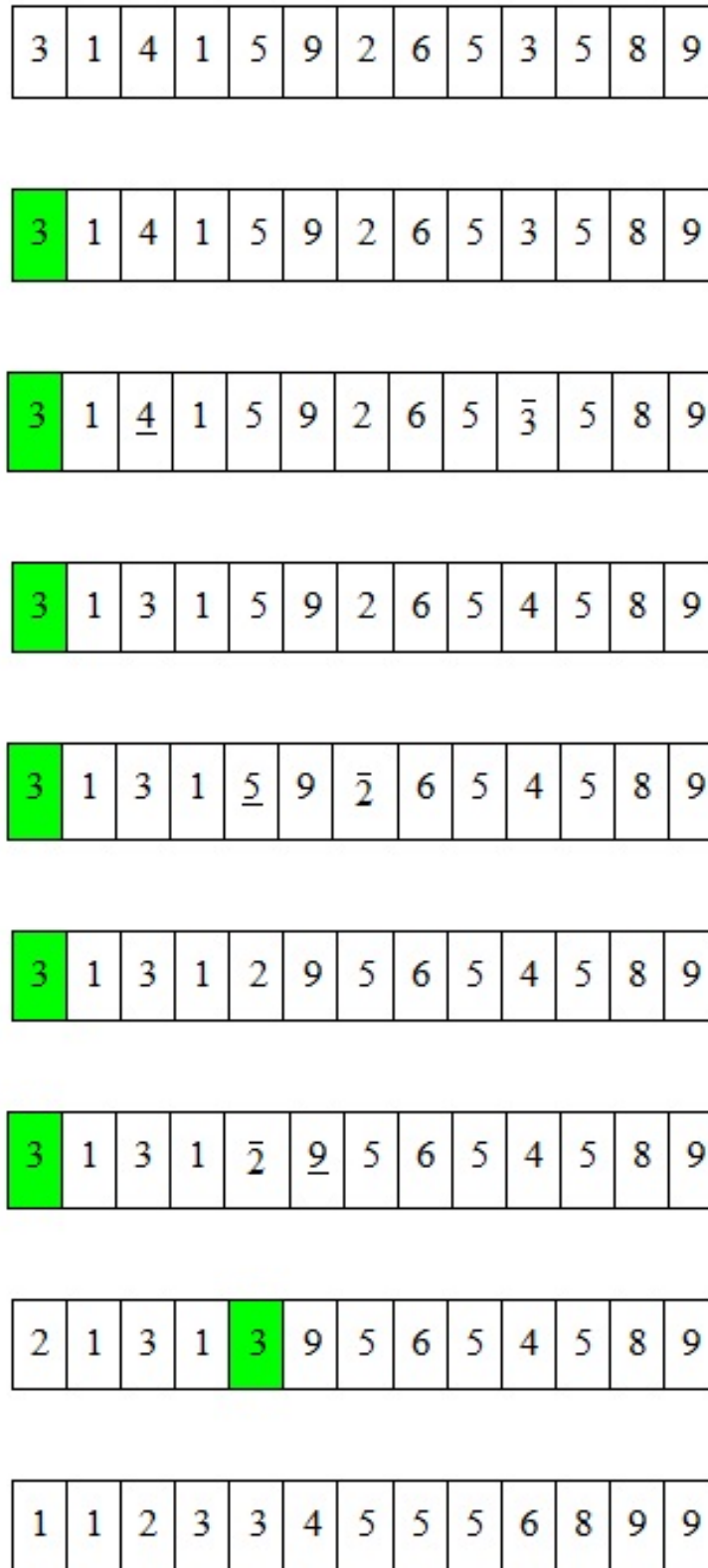


Figure 5.5

Hình 4.2

Quicksort sẽ 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 (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 $\text{pivot}(T[1..n], l)$. Theo $\text{pivot}()$ thì l nằm giữa 1 và n và xác suất là $1/n$. Thời gian để tìm vật trung tâm $g(n)$ là tuyến tính. Thời gian để dùng đệ quy 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:

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

Figure 5.6

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

$$\begin{aligned} \text{Ta có } t(n) &\leq dn + \frac{1}{n} \sum_{l=1}^n (g(n) + t(n-1) + t(n-l)) && \text{với } n > n_0 \\ &\leq dn + \frac{2}{n} \sum_{k=0}^{n-1} t(k) && \text{với } n > n_0 \end{aligned} \quad (2.4)$$

Figure 5.7

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 $O(n \log n)$.

Thật vậy ta có định lý sau:

Định lý: Quicksort cần $n \log n$ 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 $t(n) = cn \log n$ với mọi $n \geq 2$. với c là hằng số.

Dùng phương pháp qui nạp để c/m :

- Với mọi n nguyên dương: ($2 \leq n \leq n_0$)

Dễ thấy $t(n) \leq cn \log n$

$$\text{Chọn } c \geq \frac{t(n)}{n \log n}, \forall n, \quad 2 \leq n \leq n_0 \quad (2.5)$$

Figure 5.8

- Bước qui nạp

Giả thiết rằng: $t(k) \leq ck \log k$ với mọi $2 \leq k < n$

ta chỉ ra c sao cho $t(n) \leq cn \log n$

Lấy $a = t(0) + t(1)$

Từ 7.2 ta có

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

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

Figure 5.9

Theo giả thiết qui nạp : $t(k) = ck \log k$

$$\begin{aligned}
\Rightarrow \quad t(n) &\leq dn + \frac{2a}{n} + \frac{2}{n} \left(\sum_{k=2}^{n-1} ck \log k \right) \\
&\leq dn + \frac{2a}{n} + \frac{2c}{n} \int_{k=2}^n x \log x dx \\
&= dn + \frac{2a}{n} + \frac{2c}{n} \left(\frac{x^2 \log x}{2} - \frac{x^2}{4} \right) \Bigg|_{x=2}^n \\
&\leq dn + \frac{2a}{n} + \frac{2c}{n} \left(\frac{n^2 \log n}{2} - \frac{n^2}{4} \right) \\
&= dn + \frac{2a}{n} + cn \log n - \frac{cn}{2}
\end{aligned}$$

Figure 5.10

$$\begin{aligned}
&= cn \log n - \left(\frac{c}{2} - d - \frac{2a}{n^2} \right) n \\
t(n) &= cn \log n \text{ với điều kiện là } \left(\frac{c}{2} - d - \frac{2a}{n^2} \right) \geq 0, \text{ hay } c \geq 2d + 4a/n^2
\end{aligned}$$

Figure 5.11

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

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

Figure 5.12

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

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

Figure 5.13

Hay ta có $t(n) \leq cn \log n$ với mọi $n \geq 2$, 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 $O(n \log n)$. Câu hỏi đặt ra là liệu có thể sửa đổi quicksort để nó sắp xếp với thời gian $O(n \log n)$ trong trường 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 $O(n^2)$ để sắp xếp n phần tử trong trường hợp xấu nhất (khi tất cả các phần tử của mảng cần sắp là bằng nhau).

5.3 Bài toán hoán đổi³

5.3.1 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 ($n-m$ phần 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 : $(5, 6, 7, 8, 1, 2, 3, 4)$

³This content is available online at <<http://voer.edu.vn/content/m12845/1.1/>>.

5.3.2 Phân tích thuật toán

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

Nếu $m < n - m$:

- Nếu $m < n - m$: hoán đổi m phần tử đầu với m phần 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 quy để giải và quá trình gọi đệ quy sẽ dừng khi đạt tới sự hoán đổi 2 phần có độ dài bằng nhau

5.3.3 Thuật toán

// Hoán đổi m phần tử

Input : $a[1..n]$, m . ($m \leq n$)

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.

5.3.4 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:

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

Figure 5.14

$\Rightarrow T(i, j) = i + j - \text{UCLN}(i, j)$ (Ước chung lớn nhất của i, j)

Chương 6

Bài 5: Cơ bản về thuật toán Quay Lui (Back Tracking)

6.1 Sơ đồ chung của thuật toán quay lui¹

6.1.1 Ý tưởng

Nét đặc trưng của phương pháp quay lui là các bước hướng tới lời giải cuối cùng của bài toán hoàn toàn được làm thử

Tại mỗi bước, nếu có một lựa chọn thì ghi nhận lại lựa chọn này và tiến hành các bước thử tiếp theo. Còn ngược lại không có lựa chọn nào thích hợp thì quay lại bước thử trước, xóa bỏ sự ghi nhận và quay lại chu trình thử với các lựa chọn còn lại

Hành động này được gọi là quay lui, thuật toán thể hiện phương pháp này gọi là thuật toán quay lui

Điểm quan trọng của thuật toán là phải ghi nhớ tại mỗi bước đi qua để tránh trùng lặp khi quay lui. Để thấy là các thông tin này cần được lưu trữ vào một ngăn xếp, nên thuật toán thể hiện ý thiết kế một cách đệ quy.

6.1.2 Sơ đồ chung của thuật toán

Lời giải của bài toán thường biểu diễn bằng một vec tơ gồm n thành phần $x = (x_1, \dots, x_n)$ phải thỏa mãn các điều kiện nào đó. Để chỉ ra lời giải x , ta phải xây dựng dần các lời giải x_i

- Tại mỗi bước i :

+ Đã xây dựng xong các thành phần x_1, \dots, x_{i-1}

+ Xây dựng thành phần x_i bằng cách lần lượt thử tất cả các khả năng mà x_i

có thể chọn :

Nếu một khả năng j nào đó phù hợp cho x_i thì xác định x_i theo khả năng j . Thường phải có thêm thao tác ghi nhận trạng thái mới của bài toán để hỗ trợ cho bước quay lui. Nếu $i = n$ thì ta có được một lời giải, ngược lại thì tiến hành bước $i+1$ để xác định x_{i+1} .

Nếu không có một khả năng nào chấp nhận được cho x_i thì ta lùi lại bước trước (bước $i-1$) để xác định lại thành phần x_{i-1} .

Để đơn giản, ta giả định các khả năng chọn lựa cho các x_i tại mỗi bước là như nhau, do đó ta phải có thêm một thao tác kiểm tra khả năng j nào là chấp nhận được cho x_i .

Mô hình của phương pháp quay lui có thể viết bằng thủ tục sau, với n là số bước cần phải thực hiện, k là số khả năng mà x_i có thể chọn lựa.

Try(i)

¹This content is available online at <<http://voer.edu.vn/content/m12879/1.1/>>.

```

for ( j = 1 → k)
If (  $x_i$  chấp nhận được khả năng j)
{
Xác định  $x_i$  theo khả năng j;
Ghi nhận trạng thái mới;
if(  $i < n$ )
Try(i+1);
else
Ghi nhận nghiệm;
Trả lại trạng thái cũ của bài toán;
}
Ghi chú:

```

Tìm nghiệm bằng phương pháp quay lui có thể chuyển về việc tìm kiếm trên cây không gian các trạng thái, với cây được xây dựng từng mức như sau :

- Các con của gốc (thuộc mức 1) là khả năng có thể chọn cho x_1
- Giả sử x_{i-1} là một nút ở mức thứ $i-1$, khi đó các nút con của x_{i-1} là các khả năng mà x_i có thể chọn, một khi đã tìm được các thành phần x_1, \dots, x_{i-1} .

Như vậy, mỗi nút x_i của cây biểu diễn một lời giải bộ phận, đó là các nút nằm trên đường đi từ gốc đến nút đó

Ta có thể nói việc tìm kiếm nghiệm bằng phương pháp quay lui chính là tìm kiếm theo chiều sâu trên cây không gian các trạng thái.

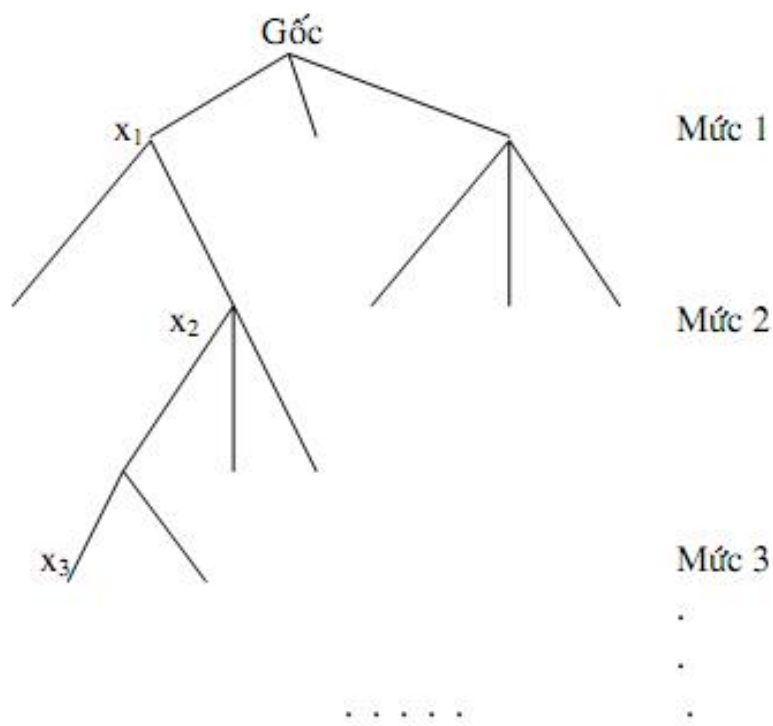


Figure 6.1

Chúng ta sẽ nghiên cứu một số các ví dụ minh họa cho thuật toán quay lui

6.2 Bài toán ngựa đi tuần²

6.2.1 Bài toán

Cho bàn cờ có $n \times n$ ô. Một con ngựa được phép đi theo luật cờ vua, đầu tiên được đặt ở ô có tọa độ x_0, y_0 . Vấn đề là hãy chỉ ra các hành trình (nếu có) của ngựa – Đó là ngựa đi qua tất cả các ô của bàn cờ, mỗi ô đi qua đúng một lần.

6.2.2 Phân tích thiết kế thuật toán

Cách giải quyết rõ ràng là xét xem có thể thực hiện một nước đi tiếp nữa hay không. Sơ đồ đầu tiên có thể phát thảo như sau:

```

Try(i)
for ( j = 1 → k)
If (  $x_i$  chấp nhận được khả năng k)
{

```

²This content is available online at <<http://voer.edu.vn/content/m12885/1.1/>>.

```

Xác định  $x_i$  theo khả năng của  $k$ ;
Ghi nhận trạng thái mới;
If ( $i < n^2$ )
Try( $i+1$ );
Else
Ghi nhận nghiệm;
Trả lại trạng thái cũ của bài toán;
}

```

Để mô tả chi tiết thuật toán, ta quy định về kiểu dữ liệu lưu trữ và các thao tác:

- Biểu diễn bàn cờ .
- Các khả năng chọn lựa cho x_i ?
- Cách thức xác định x_i theo j .
- Cách thức ghi trạng thái mới, trả về trạng thái cũ
- Ghi nhận nghiệm. . .

* Ta sẽ biểu diễn bàn cờ bằng 1 ma trận vuông cấp n : $\text{int } h[n][n]$; Sở dĩ thể hiện mỗi ô cờ bằng 1 số nguyên thay cho giá trị boole (để đánh dấu ô đã được đi qua chưa) là vì ta muốn lần dò theo quá trình dịch chuyển của con ngựa

Ta qui ước như sau :

$h[x][y] = 0 \equiv \hat{O} \langle x, y \rangle$ ngựa chưa đi qua;

$h[x][y] = I \equiv \hat{O} \langle x, y \rangle$ ngựa đã đi qua ở bước thứ i ($1 \leq i \leq n^2$).

* Các khả năng chọn lựa cho x_i ? Đó chính là các nước đi của ngựa mà x_i có thể chấp nhận được.

Với cặp tọa độ bắt đầu $\langle x, y \rangle$ như trong hình vẽ, có tất cả 8 ô $\langle u, v \rangle$ mà con ngựa có thể đi đến. Giả sử chúng được đánh số từ 0 đến 7 như hình sau:

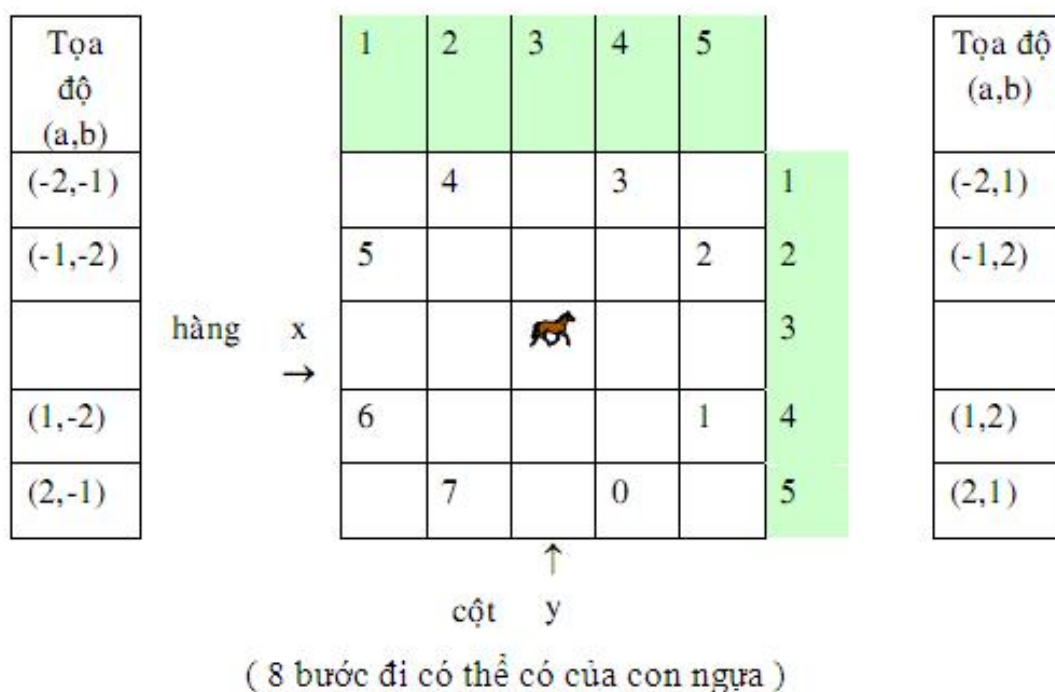


Figure 6.2

Một phương pháp đơn giản để có được u, v từ x, y là ta dùng 2 mảng a và b lưu trữ các sai biệt về tọa độ. Nếu ta dùng một chỉ số k để đánh số "bước đi tiếp" thì chi tiết đó được thể hiện bởi : $u = x + a[k]; v = y + b[k]; k = 0, 7$.

Điều kiện "chấp nhận được" có thể được biểu diễn kết hợp của các điều kiện:

- Ô mới phải thuộc bàn cờ ($1 \leq u \leq n$ và $1 \leq v \leq n$) và chưa đi qua ô đó, nghĩa là $h[u, v] = 0$;

* Để ghi nhận nước đi hợp lệ ở bước i , ta gán $h[u][v] = i$; và để hủy một nước đi thì ta gán $h[u][v] = 0$.

* Ma trận h ghi nhận kết quả nghiệm. Nếu có $\langle x, y \rangle$ sao cho $h[\langle x, y \rangle] = 0$ thì đó không phải là lời giải của bài toán, còn ngược lại h chứa đường đi của ngựa. Vậy thuật toán có thể mô tả như sau :

Input n , //Kích thước bàn cờ

x, y ; //Tọa độ xuất phát ở bước i

Output h ;

Mô tả :

Try(i, x, y)

for($k = 0; k \leq 7; k++$)

{

$u = x + a[k];$

$v = y + b[k];$

if ($1 \leq u, v \leq n$ && $h[u][v] == 0$)

{

```

h[u][v] = i;
if (i < n*n) Try(i+1,u,v);
else
    xuất_h(); // In ma trận h
}
h[u][v] = 0;
}
}

```

Thủ tục này xuất tất cả các lời giải, nếu có.

Thủ tục đệ qui được khởi động bằng một lệnh gọi các tọa độ đầu x_0, y_0 là tham số. Ô xuất phát có trị 1, còn các ô khác được đánh dấu còn trống.

```
H[x0][y0] = 1; Try(2,x , y );
```

Các mảng a và b có thể khởi đầu như sau :

```
int a[8]= {2,1,-1,-2,-2,-1,1,2};
```

```
int b[8]= {1,2,2,1,-1,-2,-2,-1};
```

* Các lời giải sau là một số kết quả cho từ thuật toán trên :

	n=5	x=1	y=1	
1	6	15	10	21
14	9	20	5	16
19	2	7	22	11
8	13	24	17	4
25	18	3	12	23

	n=6	x=2	y=3		
36	17	6	29	8	11
19	30	1	10	5	28
16	35	18	7	12	9
23	20	31	2	27	4
34	15	22	25	32	13
21	24	33	14	3	26

Figure 6.3

* Với $n = 5$, các tọa độ xuất phát sau không có lời giải : (2,3), (3,2)...

6.3 Bài toán 8 quân hậu³

6.3.1 Bài toán

Chúng ta cần đặt 8 con hậu vào bàn cờ $8 * 8$ sao cho chúng không tấn công nhau, tức là không có cặp con hậu nào nằm cùng hàng, cùng cột, cùng đường chéo.

Bài toán này là một ví dụ nổi tiếng về việc dùng các phương pháp thử và sai và phương pháp quay lui.

6.3.2 Phân tích và thiết kế thuật toán

Mấu chốt của thuật toán rõ ràng là xét xem có thể đặt quân hậu tiếp theo như thế nào. Theo luật cờ vua, một quân hậu có thể ăn các quân khác nếu nằm trên cùng 1 đường, đường này có thể là :

³This content is available online at <<http://voer.edu.vn/content/m12876/1.1/>>.

- Hàng,
- Cột,
- Các đường chéo (đi qua tọa độ vị trí của hậu).

Suy ra rằng mỗi hàng chỉ có thể chứa 1 và chỉ 1 quân hậu. Nên việc chọn vị trí cho quân hậu thứ i có thể giới hạn được ở hàng thứ i . Như thế tham số i trở thành chỉ hàng, và quá trình chọn vị trí cho quân hậu tiến hành trên toàn giá trị có thể có của các cột j .

Ta quy ước :

$x[i]$ // Chỉ quân hậu thứ i : nằm ở hàng i .

$x[i] = j$ // quân hậu thứ i đặt ở cột j ;

Để quân hậu i (trên hàng i) chấp nhận cột j thì cột j và 2 đường chéo qua ô $\langle i, j \rangle$ phải còn trống (tức là không có quân hậu khác chiếm lĩnh)

Lưu ý rằng trong 2 đường chéo :

- Đường chéo ngược (vuông góc với đường chéo chính) : tất cả các ô đều có tổng 2 tọa độ i và j là hằng số;

- Đường chéo thuận (song song với đường chéo chính) : gồm tất cả các ô $\langle i, j \rangle$ mà có hiệu các tọa độ $(i - j)$ là hằng số.

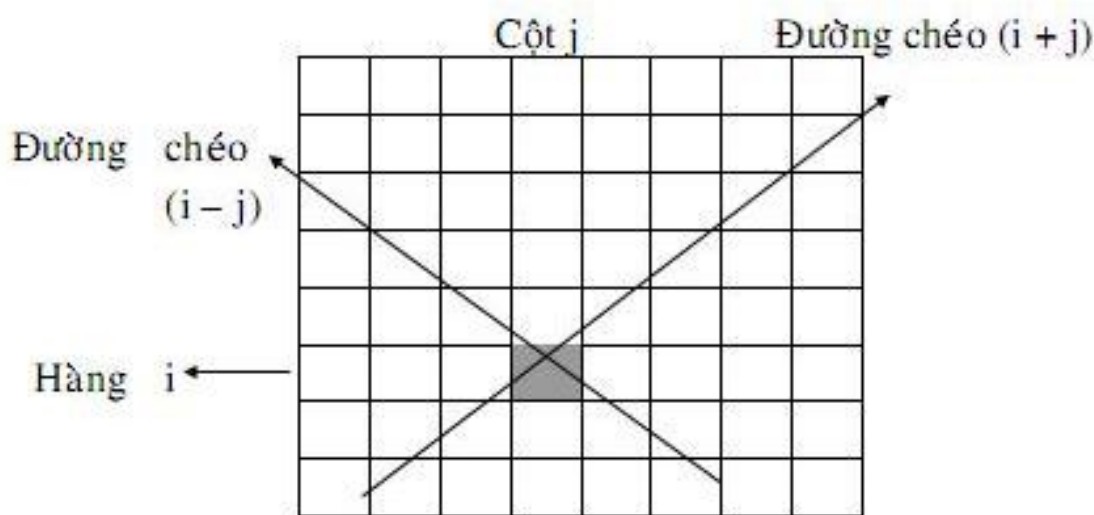


Figure 6.4

Do đó ta sẽ chọn các mảng Boole 1 chiều để biểu diễn các trạng thái này :

$a[j] = 1$: Có nghĩa là không có quân hậu nào ở cột.

$b[i+j] = 1$: Có nghĩa là không có quân hậu nào ở đường chéo ngược $(i+j)$.

$c[i - j] = 1$: Có nghĩa là không có quân hậu nào ở đường chéo thuận $(i - j)$.

Vì :

$1 \leq i, j \leq 8 \rightarrow 2 \leq i+j \leq 16$ Và $-7 \leq i - j \leq 7$.

Nên ta có thể khai báo :

int $x[8]$,

a[8],

```
b[15],
```

```
c[15];
```

Với các dữ liệu đã cho, thì lệnh đặt quân hậu sẽ thể hiện bởi :

```
x[ i ] = j; // đặt quân hậu thứ i trên cột j.
```

```
a[ j ] = 0; // Khi đặt hậu tại cột j, thì cột j không còn trống nữa
```

```
b[ i+ j ] = 0; // Các đường chéo tương ứng cũng không còn c[ i - j ] = 0; // trống nữa .
```

Còn lệnh Dời quân hậu là :

```
// Làm cho hàng i và các đường chéo tương ứng trở thành trống
```

```
a[ j ] = 1;
```

```
b[ i+ j ] = 1;
```

```
c[ i - j ] = 1;
```

Còn điều kiện an toàn là ô có tọa độ (i, j) nằm ở hàng và các đường chéo chưa bị chiếm (được thể hiện bằng giá trị True). Do đó, có thể được thể hiện bởi biểu thức logic : $a[j] \&\& b[i+j] \&\& c[i-j]$

```
Try (i)
```

```
{
```

```
for (j = 1; j <= 8; j++)
```

```
if (a[j] && b[i+j] && c[i-j])
```

```
{
```

```
x[i] = j; a[j] = 0; b[i+j] = 0;
```

```
c[i-j] = 0;
```

```
if (i < 8 )
```

```
try (i+1);
```

```
else
```

```
Xuất(x);
```

/* Sau khi in 1 lời giải xong, trả lại tình trạng ban đầu còn trống cho hàng a[j], đường chéo i+j và đường chéo i-j, để tìm lời giải khác */

```
a[ j ] = 1;
```

```
b[i+j] = 1;
```

```
c[i-j] = 1;
```

```
}
```

```
}
```

Ghi chú :

Thuật toán này tìm được tất cả 92 lời giải. Thực ra là chỉ có 12 lời giải khác nhau thật sự, đó là vì thuật toán không ghi nhận tính đối xứng.

6.4 Bài toán tìm kiếm đường đi trên đồ thị⁴

$G = (V, U)$ là đơn đồ thị (có hướng hoặc vô hướng). $V = \{1, \dots, n\}$ là tập các đỉnh, U là tập cạnh (cung). Với $s, t \in V$, tìm tất cả các đường đi từ s đến t .

Các thuật toán tìm kiếm cơ bản :

Thuật toán DFS : Tìm kiếm theo chiều sâu.

Thuật toán BFS : Tìm kiếm theo chiều rộng.

6.4.1 Thuật toán DFS (Depth First Search)

6.4.1.1 Ý tưởng

Thuật toán DFS tiến hành tìm kiếm trong đồ thị theo chiều sâu. Thuật toán thực hiện việc thăm tất cả các đỉnh có thể đạt được cho tới đỉnh t từ đỉnh s cho trước. Đỉnh được thăm càng muộn sẽ càng sớm được duyệt xong (cơ chế LIFO - Vào Sau Ra Trước). Nên thuật toán có thể tổ chức bằng một thủ tục đệ quy quay lui

⁴This content is available online at <<http://voer.edu.vn/content/m12893/1.1/>>.

6.4.1.2 Mô tả thuật toán

Input $G = (V, U)$, s, t

Output Tất cả các đường đi từ s đến t (nếu có). DFS (int s) ?

for ($u = 1$; $u \leq n$; $u++$)

{

if (chấp nhận được)

{

Ghi nhận nó;

if ($u \neq t$) DFS(u);

else In đường đi;

bỏ việc ghi nhận;

}

}

6.4.2 Thuật toán BFS (Breadth First Search)

6.4.2.1 Ý tưởng

Thuật toán BFS tiến hành tìm kiếm trên đồ thị theo chiều rộng. Thuật toán thực hiện việc thăm tất cả các đỉnh có thể đạt được cho tới đỉnh t từ đỉnh s cho trước theo từng mức kề. Đỉnh được thăm càng sớm thì sẽ càng sớm được duyệt xong (cơ chế FIFO - Vào Trước Ra Trước).

6.4.2.2 Mô tả thuật toán

Input $G = (V, E)$, $s, t \in V$;

Output

Đường đi từ s đến t .

Mô tả :

- . . .

Thuật toán có không quá n bước lặp; một trong hai trường hợp sau xảy ra :

- Nếu với mọi $i, t \neq A_i$: không có đường đi từ s đến t ;

- Ngược lại, $t \in A(m)$ với m nào đó. Khi đó tồn tại đường đi từ s tới t , và đó là một đường đi ngắn nhất từ s đến t .

Trong trường hợp này, ta xác định được các đỉnh trên đường đi bằng cách

quay ngược lại từ t đến các đỉnh trước t trong từng các tập trước cho đến khi gặp s .

Mình hoạ :

Cho đơn đồ thị có hướng :

6.5 Một số bài toán khác⁵

6.5.1 Bài toán liệt kê các dãy nhị phân độ dài n

6.5.1.1 Bài toán:

Liệt kê các dãy có chiều dài n dưới dạng $x_1x_2...x_n$, trong đó $x_i \in \{ 0,1 \}$.

⁵This content is available online at <<http://voer.edu.vn/content/m12899/1.1/>>.

6.5.1.2 Phân tích, thiết kế thuật toán:

Ta có thể sử dụng sơ đồ tìm tất cả các lời giải của bài toán. Hàm Try(i) xác định xi, trong đó xi chỉ có 1 trong 2 giá trị là 0 hay 1. Các giá trị này mặc nhiên được chấp nhận mà không cần phải thoả mãn điều kiện gì. Nên Hàm try(i) có thể viết như sau :

```

Try ( i) ≡
{
for (j = 0; j <= 1; j++)
{
x[i] = j;
if (i < n )
Try (i+1);
else
Xuất(x);
}
}

```

Cây không gian các trạng thái của bài toán có thể mô tả bởi

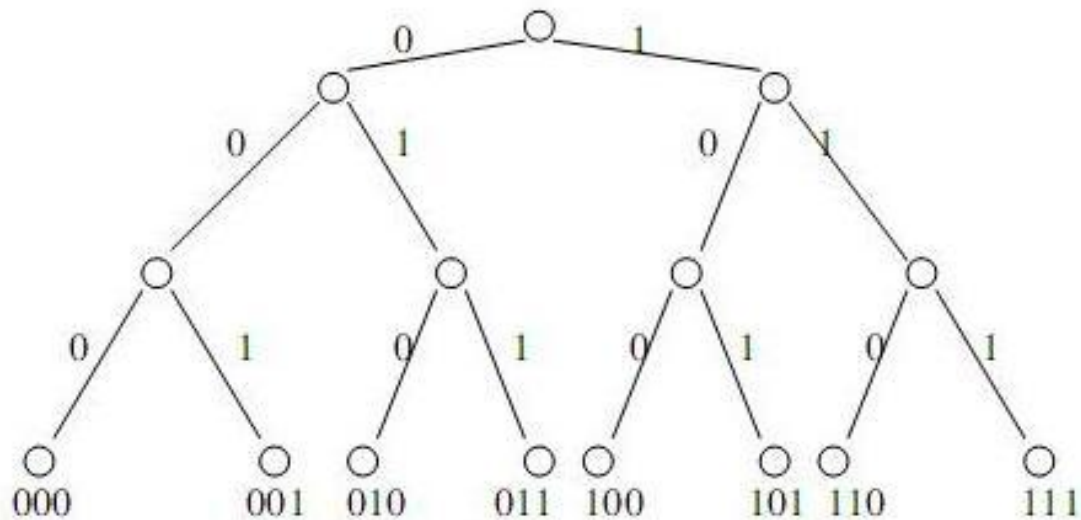


Figure 6.5

6.5.2 Bài toán liệt kê các hoán vị**6.5.2.1 Bài toán:**

Liệt kê hoán vị của n số nguyên dương đầu tiên

6.5.2.2 Phân tích, thiết kế thuật toán:

Ta biểu diễn các hoán vị dưới dạng $a_1 \dots a_n$; $a_i \in \{1, \dots, n\}$ và $a_i \neq a_j$ nếu $i \neq j$. Với mọi i , a_i chấp nhận giá trị j nếu j chưa được sử dụng, và vì vậy ta cần ghi nhớ j đã được sử dụng hay chưa khi quay lui. Để làm điều này ta dùng một dãy các biến logic b_j với quy ước :

$$\forall j = \overline{1, n}: b_j = \begin{cases} 1; & \text{nếu } j \text{ chưa sử dụng} \\ 0; & \text{nếu ngược lại.} \end{cases}$$

Figure 6.6

Sau khi gán j cho a_i , ta cần ghi nhớ cho b_j ($b_j = 0$) và phải trả lại trạng thái cũ cho b_j ($b_j = \text{True}$) khi thực hiện việc in xong một hoán vị. Ta chú ý rằng dãy các biến b_j sẽ được khởi động bằng 1

Thuật toán có thể viết như sau :

```
Try( i )
{
  for ( j = 1; j <= n; j++)
    if ( b[j] )
    {
      a[i] = j;
      b[j] = 0; // Ghi nhận trạng thái mới
      if ( i < n )
        Try(i+1);
      else
        Xuất();
      b[j] = True; // Trả lại trạng thái cũ
    }
}
```

6.5.3 Bài toán liệt kê tổ hợp

6.5.3.1 Bài toán:

Liệt kê các tổ hợp chập k của n phần tử

6.5.3.2 Phân tích, thiết kế thuật toán:

Ta sẽ biểu diễn tổ hợp dưới dạng $x_1 \dots x_k$; Trong đó :

$$1 \leq x_1, x_2, \dots, x_k \leq n$$

Ta nhận xét rằng với mọi $j \in \{1, \dots, n\}$: x_i chấp nhận $j \equiv i \in \{c_{i-1}+1, \dots, n-k+i\}$. Các giá trị j thỏa điều kiện trên mặc nhiên được chấp nhận, nên ta không cần dùng các biến boole để ghi nhớ nữa.

Thuật toán có thể viết như sau :

```
Try( i )
for ( j = 1; j <= n; j++)
  if( x[i-1] + 1 <= j <= n - k + i )
  {
```

```

x[i] = j;
if (i < k) Try(i+1);
else
  Xuất(x);
}

```

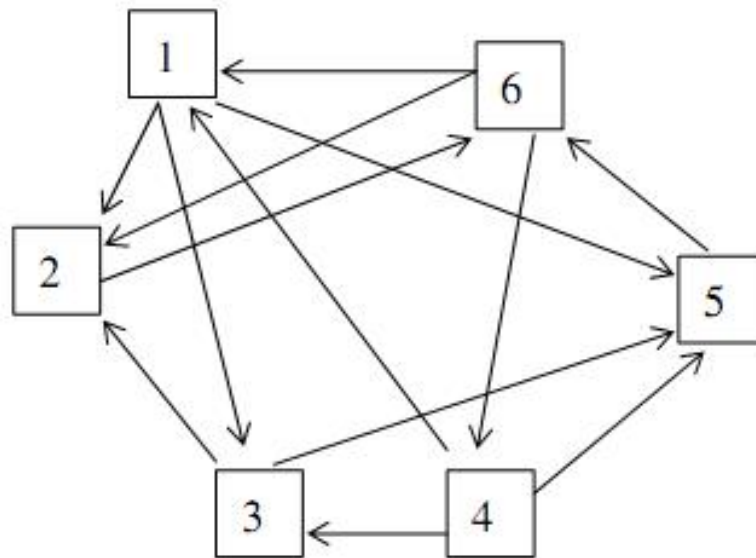


Figure 6.7

Tìm đường đi từ đỉnh (1) đến đỉnh (4) : $A(0) = \{1\}$;
 $A(1) = \{2,3,5\}$ $A(2) = \{6\}$ $A(3) = \{4\}$
 Đường đi ngắn nhất tìm được là 4 ? 6 ? 5 ? 1 , có chiều dài là 3.

Chương 7

Bài 6: Cơ bản về thuật toán nhánh cận và các bài toán

7.1 Sơ đồ chung của thuật toán¹

7.1.1 Ý tưởng:

Với các bài toán tìm phương án tối ưu, nếu chúng ta xét hết tất cả các phương án thì mất rất nhiều thời gian, nhưng nếu sử dụng phương pháp tham ăn thì phương án tìm được chưa hẳn đã là phương án tối ưu. Nhánh cận là kỹ thuật xây dựng cây tìm kiếm phương án tối ưu, nhưng không xây dựng toàn bộ cây mà sử dụng giá trị cận để hạn chế bớt các nhánh. Cây tìm kiếm phương án có nút gốc biểu diễn cho tập tất cả các phương án có thể có, mỗi nút lá biểu diễn cho một phương án nào đó. Nút n có các nút con tương ứng với các khả năng có thể lựa chọn tập phương án xuất phát từ n. Kỹ thuật này gọi là phân nhánh. Ý tưởng chính của nó như sau :

Trong quá trình duyệt ta luôn giữ lại một phương án mẫu (có thể xem là lời giải tối ưu cục bộ - chẳng hạn có giá nhỏ nhất tại thời điểm đó). Đánh giá nhánh cận là phương pháp tính giá của phương án ngay trong quá trình xây dựng các thành phần của phương án theo hướng đang xây dựng có thể tốt hơn phương án mẫu hay không. Nếu không ta lựa chọn theo hướng khác.

Với mỗi nút trên cây ta sẽ xác định một giá trị cận. Giá trị cận là một giá trị gần với giá của các phương án. Với bài toán tìm **min** ta sẽ xác định **cận dưới** còn với bài toán tìm **max** ta sẽ xác định **cận trên**. Cận dưới là giá trị nhỏ hơn hoặc bằng giá của phương án, ngược lại cận trên là giá trị lớn hơn hoặc bằng giá của phương án.

7.1.2 Mô hình chung

Giả sử bài toán tối ưu cho là :

Tìm $\min\{f(x) : x \in D\}$;

¹This content is available online at <<http://voer.edu.vn/content/m12922/1.1/>>.

$$\begin{aligned} & \text{Tìm } \min \{f(x) : x \in D\}; \\ & \text{Với } X = \left\{ a = (a_1, \dots, a_n) \in \prod_{i=1}^n A_i : P(x) \right\}; \quad |A_i| < \infty; \forall i = \overline{1, n}. \quad P \text{ là một tính} \\ & \text{chất trên } \prod_{i=1}^n A_i. \end{aligned}$$

Figure 7.1

Nghiệm của bài toán nếu có sẽ được biểu diễn dưới dạng $x = (x_1, \dots, x_n)$. Trong quá trình liệt kê theo phương pháp quay lui, ta xây dựng dần các thành phần của nghiệm.

Một bộ phận i thành phần (x_1, \dots, x_i) sẽ gọi là một lời giải (phương án) bộ phận cấp i . Ta gọi X_i là tập các lời giải bộ phận cấp i , mọi $i = \overline{1, n}$.

Đánh giá cận là tìm một hàm g xác định trên các X_i sao cho :

$$g(x_1, \dots, x_i) \leq \min \{f(a) : a = (a_1, \dots, a_n) \in X, x_i = a_i, \text{ mọi } i = \overline{1, n}\}.$$

Thủ tục quay lui sửa lại thành thủ tục nhánh cận như sau :

```

Try (i)
for (j = 1 -> n)
if( Chấp nhận được )
{
Xác định  $x_i$  theo  $j$ ;
Ghi nhận trạng thái mới;
If ( i == n )
Cập nhật lời giải tối ưu ;
else
{
Xác định cận  $g(x_1, \dots, x_i)$ 
If  $g(x_1, \dots, x_i) \leq f^*$ 
Try (i+1);
}
} // Trả bài toán về trạng thái cũ

```

7.2 Bài toán người du lịch²

7.2.1 Bài toán:

Một người du lịch muốn tham quan n thành phố T_1, \dots, T_n . Xuất phát từ một thành phố nào đó người du lịch muốn đến tất cả các thành phố còn lại, mỗi thành phố đi qua đúng 1 lần rồi quay trở lại thành phố xuất phát.

Gọi C_{ij} là chi phí đi từ thành phố T_i đến thành phố T_j . Hãy tìm một hành trình thỏa yêu cầu bài toán sao cho chi phí là nhỏ nhất.

7.2.2 Phân tích, thiết kế thuật toán

Gọi Π là một hoán vị của $\{1, \dots, n\}$ thì một thành phố thỏa mãn yêu cầu bài toán có dạng : $T_{\Pi(1)} \rightarrow T_{\Pi(2)} \rightarrow \dots \rightarrow T_{\Pi(n)}$.

²This content is available online at <<http://voer.edu.vn/content/m12918/1.1/>>.

Nếu ta cố định một thành phố xuất phát, chẳng hạn T1, thì có $(n-1)!$ Hành trình

Bài toán chuyển về dạng :

Tìm $\text{Min}\{f(a_2, \dots, a_n) : (a_2, \dots, a_n) \text{ là hoán vị của } \{2, \dots, n\}\}.$

Với $f(a_1, \dots, a_n) = C_{1, a_2} + C_{a_2, a_3} + \dots + C_{a_{n-1}, a_n} + C_{a_n, 1}$

Cách giải bài toán sẽ kết hợp đánh giá nhánh cận trong quá trình liệt kê phương án của thuật toán quay lui.

Thiết kế thuật toán:

Input $C = (C_{ij})$

Output - $x^* = (x_1, \dots, x_n)$ // Hành trình tối ưu

- $f^* = f(x^*)$ // Giá trị tối ưu

Try (i)

for (j = 1 -> n)

if(Chấp nhận được)

{

Xác định x_i theo j;

Ghi nhận trạng thái mới;

if(i == n)

Cập nhật lời giải tối ưu;

else

{

Xác định cận $g(x_1, \dots, x_i)$

If $g(x_1, \dots, x_i) \leq f^*$)

Try (i+1);

}

// Trả bài toán về trạng thái cũ

}

o Nếu ta cố định xuất phát từ T1, ta duyệt vòng lặp từ j = 2.

o Đánh giá nhánh cận :

Đặt : $C_{\text{Min}} = \text{Min}\{C_{ij} : i, j \in \{1, \dots, n\}\}$

Giả sử vào bước i ta tìm được lời giải bộ phận cấp i là (x_1, \dots, x_i) , tức là đã đi qua đoạn đường $T_1 \rightarrow T_2 \rightarrow \dots \rightarrow T_i$, tương ứng với chi phí :

$S_i = C_{1, x_2} + C_{x_2, x_3} + \dots + C_{x_{n-1}, x_n} + C_{x_n, 1}$

Để phát triển hành trình bộ phận này thành một hành trình đầy đủ, ta còn phải đi qua $n-i+1$ đoạn đường nữa, gồm $n-i$ thành phố còn lại và đoạn quay lại T1. Do chi phí mỗi một trong $n-i+1$ đoạn còn lại không nhỏ hơn C_{Min} , nên hàm đánh giá cận có thể xác định như sau :

$g(x_1, \dots, x_i) = S_i + (n - i + 1)C_{\text{Min}}$

o Điều kiện chấp nhận được của j là thành phố Tj chưa đi qua.

Ta dùng một mảng logic Daxet[] để biểu diễn trạng thái này

$\text{Daxet}[j] = ? 1 ; T_j \text{ đã được đi qua}$

0 ; T_j chưa được đi qua

Mảng Daxet[] phải được bằng 0 tất cả.

o Xác định x_i theo j bằng câu lệnh gán : $x_i = j$

Cập nhật trạng thái mới : $\text{Daxet}[j] = 1.$

Cập nhật lại chi phí sau khi tìm được x_i : $S = S + C$

o Cập nhật lời giải tối ưu :

Tính chi phí hành trình vừa tìm được :

$\text{Tong} = S + C_{x_n, 1}$;

Nếu $(\text{Tong} < f^*)$ thì

$\text{Lgtu} = x;$

$f^* = \text{Tong};$

o Thao tác huỷ bỏ trạng thái : $\text{Daxet}[j] = 0.$

Trả lại chi phí cũ : $S = S - C_{x-1xi}$
 Thủ tục nhánh cận viết lại như sau :
 Try(i)
 for (j = 2 -> n)
 if(!Daxet[j])
 {
 $x[i] = j$; Daxet[j] = 1;
 $S = S + C[x[i-1]][x[i]]$;
 if(i==n)
 {
 //Cập nhật tối ưu
 $Tong = S + C[x[n]][x[1]]$;
 if(Tong < f*)
 {
 $Lgtu = x$;
 $f^* = Tong$;
 }
 }
 else
 {
 $g = S + (n-i+1)*Cmin$; //Danh gia cận
 if (g < f*)
 Try(i+1);
 }
 $S = S - C[x[i-1]][x[i]]$;
 Daxet[j] = 0;
 }
 Minh họa
 Ma trận chi phí:

$$C = \begin{bmatrix} \infty & 3 & 14 & 18 & 15 \\ 3 & \infty & 4 & 22 & 20 \\ 17 & 9 & \infty & 16 & 4 \\ 6 & 2 & 7 & \infty & 12 \\ 9 & 15 & 11 & 5 & \infty \end{bmatrix}$$

Figure 7.2

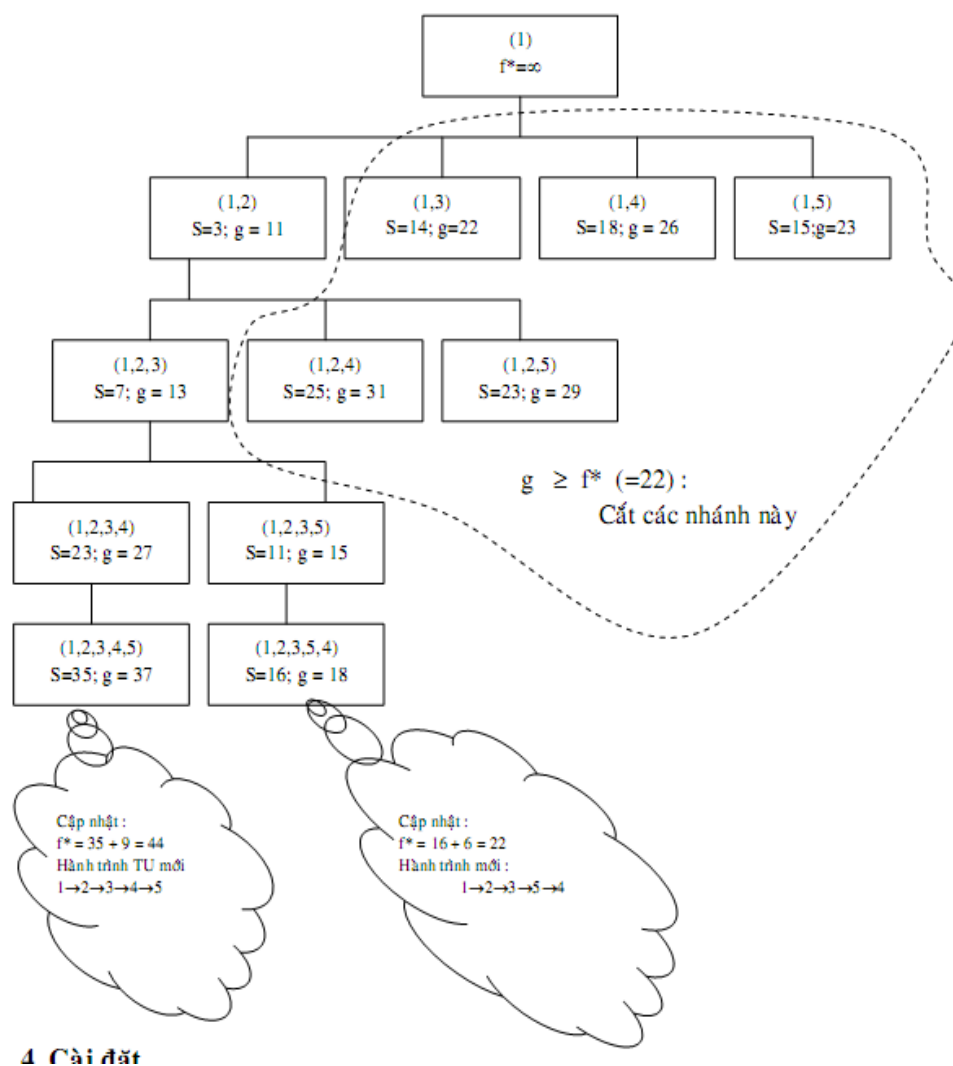


Figure 7.3

7.3 Bài toán cái túi xách³

7.3.1 Bài toán

Có n loại đồ vật, mỗi loại có số lượng không hạn chế. Đồ vật loại i , đặc trưng bởi trọng lượng W_i và giá trị sử dụng V_i , với mọi $i \in \{1, \dots, n\}$.

Cần chọn các vật này đặt vào một chiếc túi xách có giới hạn trọng lượng m , sao cho tổng giá trị sử dụng các vật được chọn là lớn nhất.

³This content is available online at <<http://voer.edu.vn/content/m12931/1.1/>>.

7.3.2 Phân tích và thiết kế thuật toán :

$$(u_1, \dots, u_n) \mapsto f(u_1, \dots, u_n) = \sum_{i=1}^n u_i v_i ; (u_1, \dots, u_n) \in D$$

Figure 7.4

$$\text{Đặt : } D = \left\{ u = (u_1, \dots, u_n) \in N^n : \sum_{i=1}^n u_i w_i \leq m \right\}$$

và : $f : D \rightarrow R^+$

Figure 7.5

Bài toán chiếc túi xách chuyển về bài toán sau :

$$\text{Tìm } x^* \in D : f^* = f(x^*) = \{f(u) : u \in D\}$$

Figure 7.6

Cho nên ta sẽ kết hợp đánh giá nhánh cận trong quá trình liệt kê các lời giải theo phương pháp quay lui.

Mô hình ban đầu có thể sử dụng như sau :

```

Try(i)
for(j = 1 -> t)
if(Chấp nhận được)
{
Xác định xi theo j;
Ghi nhận trạng thái mới;
if(i==n)
Cập nhật lời giải tối ưu;
else
{
Xác định cận trên g;
if( g(x1,..., xi) <= f*)
Try(i+1);
}
Trả lại trạng thái cũ cho bài toán;

```

}
o Cách chọn vật :

$$\text{Xét mảng đơn giá : } Dg = \left(\frac{v_1}{w_1}, \dots, \frac{v_n}{w_n} \right)$$

Figure 7.7

Ta chọn vật theo đơn giá giảm dần.
Không mất tính tổng quát, ta giả sử các loại vật cho theo thứ tự giảm dần của đơn giá.
o Đánh giá cận trên :
Giả sử đã tìm được lời giải bộ phận : (x_1, \dots, x_i)

$$S = \sum_{j=1}^i x_j v_j = S + x_j v_j.$$

Figure 7.8

Khi đó :
- Giá trị của túi xách thu được :
- Tương ứng với trọng lượng các vật đã được xếp vào chiếc túi :

$$m - TL = m - \sum_{j=1}^i x_j w_j .$$

Figure 7.9

$$TL = \sum_{j=1}^i x_j w_j = TL + x_i w_i .$$

Figure 7.10

- Do đó, giới hạn trọng lượng của chiếc túi còn lại là :

Ta thấy đây là một bài toán tìm max. Danh sách các đồ vật được sắp xếp **theo thứ tự giảm** của đơn giá để xét phân nhánh.

1. Nút gốc biểu diễn cho trạng thái ban đầu của ba lô, ở đó ta chưa chọn một vật nào. Tổng giá trị (TGT) được chọn TGT=0. Cận trên của nút gốc CT = W * Đơn giá lớn nhất.

2. Nút gốc sẽ có các nút con tương ứng với các khả năng chọn đồ vật có đơn giá lớn nhất. Với mỗi nút con ta tính lại các thông số:

· TGT = TGT (cũ) + số đồ vật được chọn * giá trị mỗi vật.

· W = W (cũ) - số đồ vật được chọn * trọng lượng mỗi vật.

· CT = TGT + W (mới) * Đơn giá của vật sẽ xét kế tiếp.

3. Trong các nút con, ta sẽ **ưu tiên phân nhánh cho nút con nào có cận trên lớn hơn** trước. Các con của nút này tương ứng với các khả năng chọn đồ vật có đơn giá lớn tiếp theo. Với mỗi nút ta lại phải xác định lại các thông số TGT, W, CT theo công thức đã nói trong bước 2.

4. Lặp lại bước 3 với chú ý: đối với những nút có **cận trên nhỏ hơn hoặc bằng giá lớn nhất tạm thời** của một phương án đã được tìm thấy thì ta không cần phân nhánh cho nút đó nữa.

5. Nếu tất cả các nút đều đã được phân nhánh hoặc bị cắt bỏ thì phương án có giá lớn nhất là phương án cần tìm.

Ví dụ : Với bài toán cái ba lô đã cho, sau khi tính đơn giá cho các đồ vật và sắp xếp các đồ vật theo thứ tự giảm dần của đơn giá ta được bảng sau.

Loại đồ vật	Trọng lượng	Giá trị	Đơn giá
b	10	25	2.5
a	15	30	2.0
d	4	6	1.5
c	2	2	1

Figure 7.11

Gọi x_1, x_2, x_3, x_4 là số lượng cần chọn tương ứng của các đồ vật b, a, d, c. Nút gốc A biểu diễn cho trạng thái ta chưa chọn bất cứ một đồ vật nào. Khi đó tổng giá trị TGT =0, trọng lượng của ba lô W=37 (theo đề ra) và cận trên CT = 37*2.5 = 92.5, trong đó 37 là W, 2.5 là đơn giá của đồ vật b.

Với đồ vật b, ta có 4 khả năng: chọn 3 đồ vật b ($X_1=3$), chọn 2 đồ vật b ($X_1=2$), chọn 1 đồ vật b ($X_1=1$) và không chọn đồ vật b ($X_1=0$). Ứng với 4 khả năng này, ta phân nhánh cho nút gốc A thành 4 con B, C, D

và E. Với nút con B, ta có $TGT = 0 + 3 \cdot 25 = 75$, trong đó 3 là số vật b được chọn, 25 là giá trị của mỗi đồ vật b. $W = 37 - 3 \cdot 10 = 7$, trong đó 37 là trọng lượng ban đầu của ba lô, 3 là số vật b được, 10 là trọng lượng mỗi đồ vật b. $CT = 75 + 7 \cdot 2 = 89$, trong đó 75 là TGT, 7 là trọng lượng còn lại của ba lô và 2 là đơn giá của đồ vật a. Tương tự ta tính được các thông số cho các nút C, D và E, trong đó cận trên tương ứng là 84, 79 và 74.

Trong các nút B, C, D và E thì nút B có cận trên lớn nhất nên ta sẽ phân nhánh cho nút B trước với hy vọng sẽ có được phương án tốt từ hướng này. Từ nút B ta chỉ có một nút con F duy nhất ứng với $X_2=0$ (do trọng lượng còn lại của ba lô là 7, trong khi trọng lượng của mỗi đồ vật a là 15). Sau khi xác định các thông số cho nút F ta có cận trên của F là 85.5. Ta tiếp tục phân nhánh cho nút F. Nút F có 2 con G và H tương ứng với $X_3=1$ và $X_3=0$. Sau khi xác định các thông số cho hai nút này ta thấy cận trên của G là 84 và của H là 82 nên ta tiếp tục phân nhánh cho nút G. Nút G có hai con là I và J tương ứng với $X_4=1$ và $X_4=0$. Đây là hai nút lá (biểu diễn cho phương án) vì với mỗi nút thì số các đồ vật đã được chọn xong. Trong đó nút I biểu diễn cho phương án chọn $X_1=3, X_2=0, X_3=1$ và $X_4=1$ với giá 83, trong khi nút J biểu diễn cho phương án chọn $X_1=3, X_2=0, X_3=1$ và $X_4=0$ với giá 81. Như vậy giá lớn nhất tạm thời ở đây là 83.

Quay lui lên nút H, ta thấy cận trên của H là $82 < 83$ nên cắt tỉa nút H.

Quay lui lên nút C, ta thấy cận trên của C là $84 > 83$ nên tiếp tục phân nhánh cho nút C. Nút C có hai con là K và L ứng với $X_2=1$ và $X_2=0$. Sau khi tính các thông số cho K và L ta thấy cận trên của K là 83 và của L là 75.25. Cả hai giá trị này đều không lớn hơn 83 nên cả hai nút này đều bị cắt tỉa. Cuối cùng các nút D và E cũng bị cắt tỉa. Như vậy tất cả các nút trên cây đều đã được phân nhánh hoặc bị cắt tỉa nên phương án tốt nhất tạm thời là phương án cần tìm. Theo đó ta cần chọn 3 đồ vật loại b, 1 đồ vật loại d và một đồ vật loại c với tổng giá trị là 83, tổng trọng lượng là 36.

Chương 8

Bài 7: Cơ bản về thuật toán Tham Lam

8.1 Thuật toán Tham Lam¹

Đặc trưng của chiến lược tham lam
Sơ đồ thuật toán

8.1.1 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 lý 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.

8.1.2 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 thỏa 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 thỏa mãn điều kiện .

¹This content is available online at <<http://voer.edu.vn/content/m12935/1.1/>>.

- Một hàm quan trọng nữa : $\text{Select}(C)$ 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 \rightarrow dựa trên căn cứ vào ảnh hưởng 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 $\rightarrow \text{Feasible}(S \cup x)$.

```
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;
}
```

8.2 Bài toán người du lịch²

8.2.1 Bài toán

Một người du lịch muốn tham quan n thành phố T_1, \dots, T_n . 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.

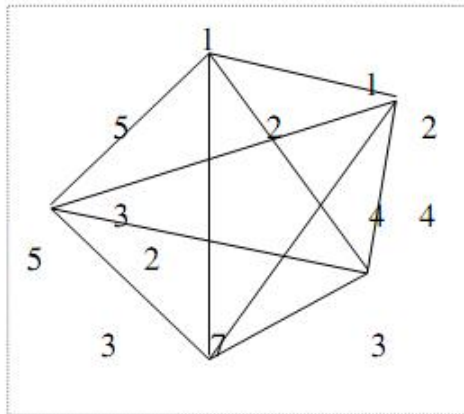
Gọi C_{ij} là chi phí đi từ thành phố T_i đến T_j . Hãy tìm một hành trình thỏa yêu cầu bài toán sao cho chi phí là nhỏ nhất.

8.2.2 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 phí tương ứng
TOUR := 0; COST := 0; v := u; // Khởi tạo
Mọi k := 1  $\rightarrow$  n ://Thăm tất cả các thành phố
// Chọn cạnh kề )
- Chọn  $\langle v, w \rangle$  là đoạn nối 2 thành phố có chi phí nhỏ nhất tính từ thành phố v đến các thành phố chưa qua.
- TOUR := TOUR +  $\langle v, w \rangle$ ; //Cập nhật lời giải
- COST := COST +  $C_{vw}$ ; //Cập nhật chi phí
// Chuyển đi hoàn thành TOUR := TOUR +  $\langle v, u \rangle$ ; COST := COST +  $C_{vw}$ 
Minh họa:
```

²This content is available online at <http://voer.edu.vn/content/m12943/1.1/>.



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

1. TOUR := 0; COST := 0; u := 1;
 $\Rightarrow w = 2;$

2

2. TOUR := <1,2>; COST := 1; u := 2;
 $\Rightarrow w = 5;$

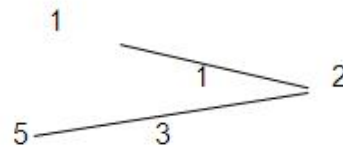
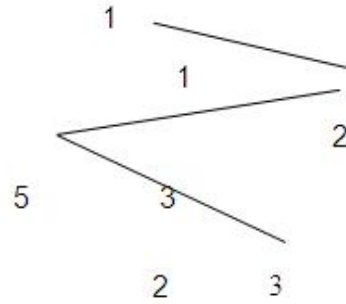


Figure 8.1

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

COST := 4; u := 5;

=> w = 3;



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

COST := 6; u := 3;

Figure 8.2

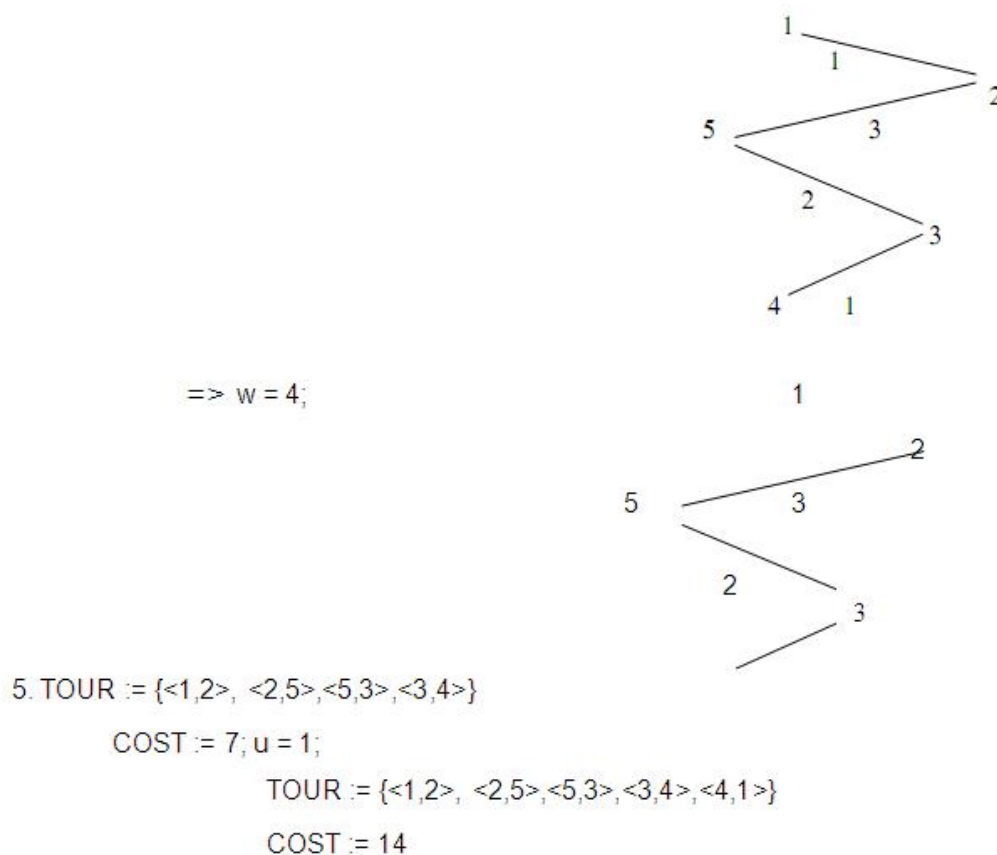


Figure 8.3

8.2.3 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 $T(n) \in O(n^2)$.

8.2.4 Cài đặt thuật toán

```
int GTS (mat a, int n, int TOUR[max], int Ddau)
{
    int v, //Đỉnh đang xét
    k, //Duyệt qua n đỉnh để chọn
    w; //Đỉnh được chọn trong mỗi bước
    int mini; //Chọn min các cạnh(cung) trong mỗi bước
    int COST; //Trong số nhỏ nhất của chu trình
    int daxet[max]; //Danh dấu các đỉnh đã được sử dụng
    for(k = 1; k <= n; k++)
        daxet[k] = 0; //Chưa đỉnh nào được xét
    COST = 0; //Lúc đầu, giá trị 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;
}

```

8.3 Thuật toán Dijkstra – Tìm đường đi ngắn nhất trong đồ thị có trọng số³

8.3.1 Bài toán

Cho $G = (V, E)$ là đơn đồ thị liên thông (vô hướng hoặc có hướng) có trọng số

$V = \{1, \dots, n\}$ là tập các đỉnh, E là tập các cạnh (cung).

Cho $s_0 \in E$. Tìm đường đi ngắn nhất đi từ s_0 đến các đỉnh còn lại. Giải bài toán trên bằng thuật toán Dijkstra.

8.3.2 Phân tích, thiết kế thuật toán

Thuật toán Dijkstra cho phép tìm đường đi ngắn nhất từ một đỉnh s đến các đỉnh còn lại của đồ thị và chiều dài (trọng số) tương ứng. Phương pháp của thuật toán là xác định tuần tự đỉnh có chiều dài đến s theo thứ tự tăng dần. Thuật toán được xây dựng trên cơ sở gán cho mỗi đỉnh các nhãn tạm thời. Nhãn tạm thời của các đỉnh cho biết cận trên của chiều dài đường đi ngắn nhất từ s đến đỉnh đó. Nhãn của các đỉnh sẽ biến đổi trong các bước lặp, mà ở mỗi bước lặp sẽ có một nhãn tạm thời trở thành chính thức. Nếu nhãn của một đỉnh nào đó trở thành chính thức thì đó cũng chính là chiều dài ngắn nhất của đường đi từ s đến đỉnh đó.

Ký hiệu :

* $L(v)$ để chỉ nhãn của đỉnh v , tức là cận trên của chiều dài đường đi ngắn nhất từ s_0 đến v .

* $d(s_0, v)$: chiều dài đường đi ngắn nhất từ s_0 đến v .

* $m(s_0, v)$ là trọng số của cung (cạnh) (s, v) .

Thuật toán Dijkstra tìm chiều dài đường đi ngắn nhất từ đỉnh s đến $n-1$ đỉnh còn lại được mô tả như sau:

³This content is available online at <<http://voer.edu.vn/content/m12950/1.1/>>.

input: G, s_0
 Output : $d(s_0, v)$, mọi $v \neq s_0$;
 Mô tả :
 o Khởi động :
 $L(v) = \infty$, mọi $v \neq s_0$; //Nhãn tạm thời
 $S = \{s_0\}$; //Tập lưu trữ các đỉnh có nhãn chính thức
 o Bước 0 :
 $d(s_0, s_0) = L(s_0) = 0$;
 $S = \{s_0\}$; // s_0 có Nhãn chính thức
 o Bước 1:
 - Tính lại nhãn tạm thời $L(v)$, v không thuộc S :
 Nếu v kề với s_0 thì
 $L(v) = \text{Min}\{L(v), L(s_0) + m(s_0, v)\}$;
 - Tìm s_1 thuộc S và kề với s_0 sao cho :

$$L(s_1) = \text{Min}\{L(v) : \forall v \notin S, \};$$

$$(\text{Khi đó : } d(s_0, s_1) = L(s_1))$$

$$- S = S \cup \{s_1\}; // S = \{s_0, s_1\}; s_1 \text{ có nhãn chính thức}$$

Figure 8.4

o Bước 2:
 - Tính lại nhãn tạm thời $L(v)$, $v \notin S$:

- Tìm $s_2 \notin S$ và kề với s_1 hoặc s_0 sao cho :
 $L(s_2) = \text{Min}\{L(v) : \forall v \notin S\}$;
 (Khi đó : $d(s, s_2) = L(s_2)$); // $0 = d(s_0, s_1) \leq d(s_0, s_2)$
 Nếu $L(s_2) = \text{Min}\{L(s_j), L(s_j) + m(s_j, s_2)\}$ thì đường đi từ s đến s_2 đi qua đỉnh s_j là ngắn nhất, và s_j là đỉnh đứng kề trước s_2 .
 $- S = S \cup \{s_2\}; // S = \{s_0, s_1, s_2\}; //s_2 \text{ có nhãn chính thức}$
 ...

Figure 8.5

Nếu v kề với s_1 thì $L(v) = \text{Min}\{L(v), L(s_1) + m(s_1, v)\}$;
 Tính chất tham lam của thuật toán Dijkstra là tại mỗi bước, chọn s_i không thuộc S và s_i là đỉnh kề với s_j , với $j = 0, i-1$ sao cho $L(s_i) = \text{Min}\{L(v) : v \notin S\}$. Minh hoạ : Xét đồ thị có hướng G :

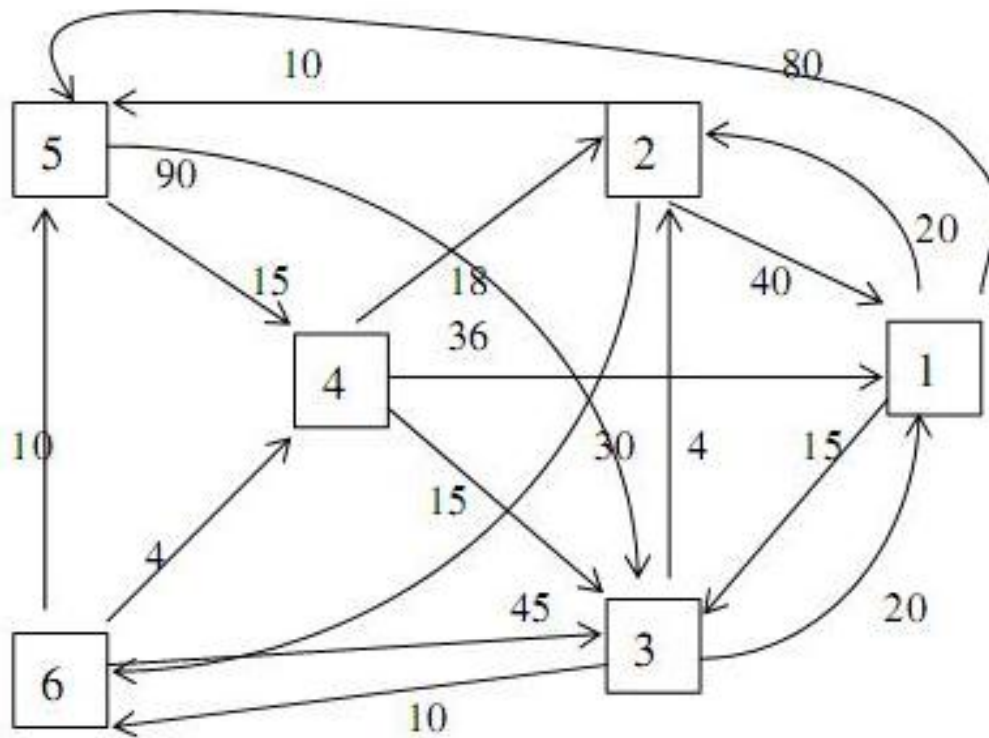


Figure 8.6

Đường đi ngắn nhất từ đỉnh $s = 1$ đến các đỉnh còn lại :
 Bảng các bước đi.

Bước lập	Đường đi ngắn nhất là đường đi từ đỉnh 1	đến đỉnh	Chiều dài của đường đi ngắn nhất từ đỉnh s (=1) đến các đỉnh khác : tsnn[]					
			1	2	3	4	5	6
Bước1	1 → 3	3	-	20	15	∞	80	∞
Bước2	1 → 3 → 2	2	-	19	-			25
Bước3	1 → 3 → 6	6	-	-	-		29	25
Bước4	1 → 3 → 6 → 4	4	-	-	-	29		-
Bước5	1 → 3 → 2 → 5	5	-	-	-	-	29	-

Figure 8.7

Chương 9

Bài 8: Cơ bản về thuật toán Quy hoạch động

9.1 Sơ đồ chung của thuật toán¹

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.

¹This content is available online at <<http://voer.edu.vn/content/m12952/1.1/>>.

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 được 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.

9.2 Bài toán thực hiện dãy phép nhân ma trận²

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

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

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 tất 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.

$$M = M_1 M_2 \dots M_n$$

Theo nhiều cách khác nhau, chẳng hạn

$$M = (\dots((M_1 M_2) M_3) \dots M_n)$$

$$= (M_1 (M_2 (M_3 \dots (M_{n-1} M_n) \dots)))$$

$$= (\dots((M_1 M_2) (M_3 M_4)) \dots), v.v. \dots$$

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 13×5 , B có kích thước 5×89 , C có kích thước 89×3 và D có kích thước 3×34 . Sử dụng cách tính

$$M = ((AB)C)D,$$

Ta phải thực hiện lần lượt tính

AB 5785 phép nhân

(AB)C 3271 phép nhân

((AB)C)D 1326 phép nhân

Và tổng cộng là 10582 phép nhân

²This content is available online at <<http://voer.edu.vn/content/m12957/1.1/>>.

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:

$$M = (M_1 M_2 \dots M_i)(M_{i+1} M_{i+2} \dots M_n)$$

Khi đó có $T(i)$ cách đặt dấu ngoặc cho thừa số thứ nhất $(M_1 M_2 \dots M_i)$ và $T(n-i)$ cách đặt dấu ngoặc cho thừa số thứ hai $(M_{i+1} M_{i+2} \dots M_n)$ và từ đó $T(i)T(n-i)$ cách tính biểu thức $(M_1 M_2 \dots M_i)(M_{i+1} M_{i+2} \dots M_n)$. 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	2674440

Figure 9.1

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 $T(n) = T(n) = 4^{nn^2}$. Như vậy, phương pháp duyệt toàn bộ không thể sử dụng để tìm cách tính hiệu quả biểu thức tích 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 đặt 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 $(M_1 M_2 \dots M_i)$ và $(M_{i+1} M_{i+2} \dots M_n)$ 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 $((M_1 M_2 \dots M_i)$ và $(M_{i+1} M_{i+2} \dots M_n)$ 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 $(M_1 M_2 \dots M_i)$ và $(M_{i+1} M_{i+2} \dots M_n)$ 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 $(M_1 M_2 \dots M_n)$ chúng ta có thể sử dụng cách tính tối ưu của hai tích con $(M_1 M_2 \dots M_i)$ và $(M_{i+1} M_{i+2} \dots M_n)$. 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

$(M_{i+1} M_{i+2} \dots M_j)$, $1 \leq i \leq j \leq n$

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 $d[0 \dots n]$, trong đó ma trận M_i có kích thước $d_{i-1} \times d_i$, $i = 1, 2, 3, \dots, n$. Ta sẽ xây dựng bảng giá trị m_{ij} lần lượt theo từng đường chéo của nó, trong đó đường chéo thứ s chứa các phần tử m_{ij} với chỉ số thoả mãn $j - i = s$. Khi đó, đường chéo $s = 0$ sẽ chứa các phần tử m_{ij} ($i = 1, 2, \dots, n$) tương ứng với tích có một phần tử M_i . Do đó, $m_{ij} = 0$, $i = 1, 2, \dots, n$. Đường chéo $s = 1$ chứa các phần tử m_{ij+1} tương ứng với tích $M_i M_{i+1}$, do ở đây không có sự lựa chọn nào khác, nên ta phải thực hiện $d_{i-1} d_i d_{i+1}$ phép nhân. Giả sử $s > 1$, khi đó đường chéo thứ s chứa các phần tử m_{ij+s} tương ứng với tích $M_i M_{i+1} \dots M_{i+s}$. Bây giờ ta có thể lựa chọn việc đặt dấu ngoặc tách đầu tiên sau một trong số các ma trận $M_i, M_{i+1}, \dots, M_{i+s-1}$. Nếu đặt dấu ngoặc đầu tiên sau M_k , $i \leq k < i+s$, ta cần thực hiện m_{ik} phép nhân để tính thừa số thứ nhất, $m_{k+1, i+s}$ phép nhân để tính thừa số thứ hai, và cuối cùng là $d_{i-1} d_k d_{i+s}$ 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 m_{ij} ta có thể sử dụng quy tắc sau đây:

$s = 0$; $m_{ij} = 0$ $i = 1, 2, \dots, n$

$s = 1$; $m_{j+1} = d_{i-1} d_i d_{i+1}$, $i = 1, 2, \dots, n-1$

$1 < s < n$; $m_{j+s} = \min\{m_{ik} + m_{k+1, i+s} + d_{i-1} d_k d_{i+s} : 1 \leq k < i+s\}$, $i = 1, 2, \dots, n-s$.

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 $s = 1$, $m_{12} = 5785$, $m_{23} = 1335$ và $m_{34} = 9078$. 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

Figure 9.2

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

Figure 9.3

Ta có số phép nhân cần thực hiện là $m_{14} = 2856$. Dấu ngoặc đầu tiên cần đặt sau vị trí $h_{14} = 3$, tức là $M = (ABC)D$. Ta tìm cách đặt dấu ngoặc đầu tiên để có m_{13} tương ứng với tích ABC. Ta có $h_{13} = 1$, 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 [U+0096] 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

$$\sum_{s=1}^{n-1} (n-s)s = n \sum_{s=1}^{n-1} s - \sum_{s=1}^{n-1} s^2$$

$$n^2(n-1)/2 - n(n-1)(2n-1)/6$$

$$(n^3 - n)/6$$

$$0 \quad (n^3)$$

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  $M_i \dots M_j$ ;
h[i,j] - ghi nhận vị trí đặt dấu ngoặc đầu tiên trong cách thực hiện nhân dãy  $M_i \dots M_j$ }
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];
 $X = Mult(i,k); \{ X = M[i] / \dots M[k] \}$ 
 $Y = Mult(k+1,j); \{ Y = M[k+1] \dots M[j] \}$ 
return  $X*Y$ ; { Nhân ma trận X và Y }
end
else
return M[i];
end;

```

Chương 10

Bài 9: Các bài toán sử dụng thuật toán Quy hoạch động

10.1 Tập độc lớn nhất trên cây¹

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 tùy ý với k con w_1, w_2, \dots, w_k . 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 w_1, w_2, \dots, w_k để tạo tập độc lập gốc tại r , 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 w_1, w_2, \dots, w_k , do và bất kỳ con w_i nào của nó không cùng chọn vào tập độc lập

¹This content is available online at <http://voer.edu.vn/content/m12965/1.1/>.

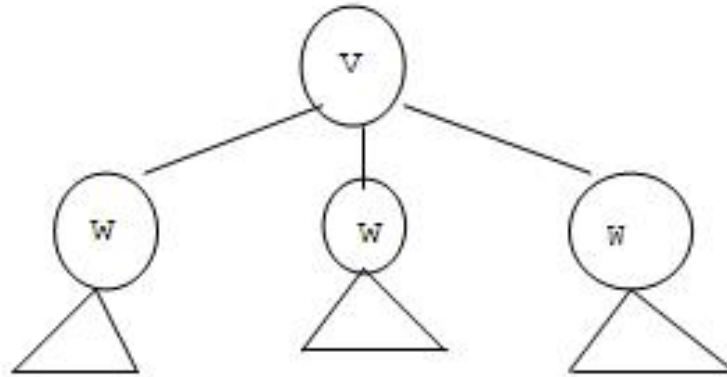


Figure 10.1

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

1. $\text{big}()$ = trọng lượng lớn nhất của tập độc lập của cây con có gốc tại .
2. $\text{bignotroot}()$ = trọng lượng lớn nhất của tập độc lập không chứa của cây con có gốc tại .

Tại đỉnh , thuật toán sẽ gọi đệ quy tính $\text{big}(w_i)$ và $\text{bignotroot}(w_i)$ với mỗi cây con gốc tại các con w_1, w_2, \dots, w_k của . Sau đó tính $\text{bignotroot}()$ và $\text{big}()$ sử dụng công thức đệ quy tương ứng với hai tình huống mô tả ở trên:

$$\begin{aligned} \text{bignotroot}(v) &= \sum_{i=1}^k \text{big}(w_i) \\ \text{big}(v) &= \max\{\text{bignotroot}(v), c(v) + \sum_{i=1}^k \text{bignotroot}(w_i)\} \end{aligned} \quad (10.1)$$

Nếu là lá thì $\text{bignotroot}() = 0$ và $\text{big}() = c()$.

Từ các phân tích trên dễ dàng xây dựng thuật toán tính $\text{big}()$, V với thời gian $O(n)$, trong đó $n = |V|$.

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 w_1, w_2, \dots, w_k 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 in Con(r) do
bignotroot := bignotroot + MaxISTree(w);
bigr := c(r);
for u in Cháu(r) do
bigr := bigr + MaxISTree(u);
end
end
end
  
```



```

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 $O(n)$.

10.2 Bài toán dãy con lớn nhất²

Trong chương 2 ta đã trình bày thuật toán chia để trị để giải bài toán dãy con lớn nhất với thời gian tính $O(n \log n)$. 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 s_i là tổng của dãy con lớn nhất trong dãy

a_1, a_2, \dots, a_i ,

$i = 1, 2, \dots, n$. Rõ ràng s_n là giá trị cần tìm.

Tổng hợp lời giải. Trước hết, ta có $s_n = a_1$. Bây giờ giả sử $i > 1$ và s_k là đã biết với $k = 1, 2, \dots, i - 1$. Ta cần tính s_i là tổng của dãy con lớn nhất của dãy con lớn nhất của dãy $a_1, a_2, \dots, a_{i-1}, a_i$.

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

- Dãy con lớn nhất của dãy a_1, a_2, \dots, a_{i-1} .
- Dãy con lớn nhất của dãy a_1, a_2, \dots, a_i kết thúc tại a_i .

Từ đó suy ra

$s_i = \max \{s_{i-1}, e_i\}$,

Trong đó e_i là tổng của dãy con lớn nhất của dãy a_1, a_2, \dots, a_i kết thúc tại a_i .

Lưu ý rằng để tính e_i , $i = 1, 2, \dots, n$, ta cũng có thể sử dụng công thức đệ quy sau:

$e_1 = a_1$;

$e_i = \max \{a_i, e_{i-1} + a_i\}$, $i > 1$.

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

```

procedure Maxsub(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;
  end;
  Để thấy thuật toán Maxsub có thời gian tính là  $O(n)$ .

```

²This content is available online at <<http://voer.edu.vn/content/m12960/1.1/>>.

10.3 Bài toán dãy con chung dài nhất³

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 $X = \langle x_1, x_2, \dots, x_m \rangle$, dãy $Z = \langle z_1, z_2, \dots, z_k \rangle$ được gọi là dãy con của dãy X nếu tìm được dãy các chỉ số $1 \leq i_1 < i_2 < \dots < i_k \leq m$ sao cho $z_j = x_{i_j}$, $j = 1, 2, \dots, k$. Chẳng hạn dãy $Z = \langle B, C, D, B \rangle$ là dãy con của dãy $X = \langle A, A, B, C, B, C, D, A, B, D, A, B \rangle$ với dãy chỉ số là $\langle 3, 4, 7, 9 \rangle$.

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 $X = \langle A, B, C, D, E, F, G \rangle$ và $Y = \langle C, C, E, D, E, G, F \rangle$ thì dãy $Z = \langle C, D, F \rangle$ là dãy con chung của hai dãy X và Y , còn dãy $\langle B, F, G \rangle$ không là dãy con chung của chúng. Dãy $\langle C, D, F \rangle$ không là dãy con chung dài nhất vì nó có độ dài 3 (số phần tử trong dãy), trong khi đó dãy $\langle C, D, E, G \rangle$ là dãy con chung của X và Y có độ dài 4. Dãy $\langle C, D, E, G \rangle$ 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 $X = \langle x_1, x_2, \dots, x_m \rangle$ và $Y = \langle y_1, y_2, \dots, y_n \rangle$. 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 $\langle i_1, i_2, \dots, i_k \rangle$ là tập con k phần tử của tập chỉ số $\{1, 2, \dots, m\}$, vì thế có tất cả 2^m 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 $0 \leq i \leq m$ và $0 \leq j \leq n$ xét bài toán $C(i, j)$; tính $C[i, j]$ là độ dài của dãy con chung dài nhất của hai dãy.

$X_i = \langle x_1, x_2, \dots, x_i \rangle$

và

$Y_j = \langle y_1, y_2, \dots, y_j \rangle$

Như vậy ta đã phân bài toán cần giải ra thành $(m+1) \times (n+1)$ 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

$c[0, j] = 0, j = 0, 1, \dots, n$ và $c[i, 0] = 0, i = 0, 1, \dots, m$.

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

Nếu $X_i = Y_j$ thì dãy con chung dài nhất của X_i và Y_j sẽ thu được bằng việc bổ sung X_i vào dãy con chung dài nhất của hai dãy X_{i-1} và Y_{j-1}

Nếu $X_i \neq Y_j$ thì dãy con chung dài nhất của X_i và Y_j sẽ là dãy con dài nhất trong hai dãy con chung dài nhất của $(X_i$ và $Y_{j-1})$ và của $(X_{i-1}$ và $Y_j)$. Từ đó ta có công thức sau để tính $C[i, j]$.

$c[i, j]$

0,

$c[i-1, j-1] + 1$

$\max\{c[i, j-1], c[i-1, j]\}$ nếu $x_i \neq y_j$ hoặc $x_i = y_j$

,

{ {

Thuật toán tìm dãy con chung dài nhất của hai dãy X và Y như sau.

Procedure LCS(X, Y);

begin

for $i := 1$ to m do $c[i, 0] := 0$;

for $j := 1$ to n do $c[0, j] := 0$;

³This content is available online at <http://voer.edu.vn/content/m12969/1.1/>.

```

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]:= [U+FOAD] ;
end
else
begin
c[i,j]:=c[i,j-1];
b[i,j]:= [U+FOAC] ;
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; (* §a ra phtö xi *)
end
else
if b[i,j]= [U+FOAD] then
PrintLCS (b,X,i-1,j)
else
Print LCS (b,X,i,j-1);
end;

```

Dễ dàng đánh giá được thời gian tính của thuật toán LCS là $O(mn)$.

Chương 11

Bài 10: Bài tập và tổng kết

11.1 Bài tập tổng kết¹

11.1.1 Thảo luận về khái niệm thuật toán

- Người học đưa ra một số bài toán trong thực tế được giải quyết như quá trình thực hiện của một thuật toán.

- Phân tích các đặc trưng của thuật toán qua các ví dụ trên.
- Chọn một bài toán để biểu diễn thuật toán bằng một số phương pháp.

11.1.2 Thảo luận về tư tưởng thiết kế thuật toán chia để trị

- Người học tìm một số bài toán trong thực tế, sau đó thảo luận phân tích thành các bài toán nhỏ theo tư tưởng của thuật toán chia để trị.

- Nhận xét công việc thực hiện các bài toán nhỏ so với bài toán ban đầu
 - Nhận xét về việc tổng hợp kết quả từ những bài toán nhỏ
- => Từ đó rút ra kết luận về sơ đồ chung của thuật toán chia để trị.

11.1.3 Thảo luận về độ phức tạp tính toán trong thuật toán chia để trị

- Phân tích độ phức tạp thuật toán của các bài toán con

- So sánh với độ phức tạp tính toán của bài toán ban đầu
- => Từ đó rút ra kết luận về ưu điểm của thuật toán chia để trị.

11.1.4 Các bài toán sử dụng phương pháp chia để trị

Các nhóm báo cáo các bài toán đã được phân công

Bài toán tìm kiếm nhị phân

* So sánh tìm kiếm tuần tự và tìm kiếm nhị phân:

- Nhắc lại phương pháp tìm kiếm tuần tự
- So sánh ưu nhược điểm của 2 phương pháp, chỉ rõ qua ví dụ cụ thể.

* Cài đặt thuật toán:

```
int tknp(int a[max],int x,int l, int r)
{
    int mid;
    if( l > r) return 0;
```

¹This content is available online at <<http://voer.edu.vn/content/m12445/1.1/>>.

```

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);
}

```

* Đánh giá độ phức tạp thuật toán:

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

=> Ta có: $T_{\text{tốt}}(n) = O(1)$

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 ($x == a[\text{giữa}]$) 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 $n \geq 2$ và $T(1) = 0$

Bài toán mảng con lớn nhất

- Phát biểu bài toán

- Tư tưởng giải thuật

- Cài đặt

- Đánh giá độ phức tạp thuật toán

Thuật toán nhân 2 ma trận

- Phát biểu bài toán

- Tư tưởng giải thuật

- Cài đặt

- Đánh giá độ phức tạp thuật toán

Thuật toán sắp xếp

- Phát biểu bài toán

- Tư tưởng giải thuật

- Cài đặt

- Đánh giá độ phức tạp thuật toán

Bài toán hoán đổi

- Phát biểu bài toán

- Tư tưởng giải thuật

- Cài đặt

- Đánh giá độ phức tạp thuật toán

11.1.5 Nhắc lại các thuật toán khác và các bài toán

Trên cơ sở lý thuyết máy Turing, ta chia được các bài toán thành 2 lớp không giao nhau : Lớp giải được bằng thuật toán, và lớp không giải được bằng thuật toán.



Figure 11.1

Đối với lớp các bài toán giải được bằng thuật toán, dựa vào các đặc trưng của quá trình thiết kế của thuật toán, ta có thể chỉ ra một số các phương pháp thiết kế thuật toán cơ bản sau đây :

a) Phương pháp chia để trị. (Divide-and-Conquer method).

Ý tưởng là : Chia dữ liệu thành từng miền đủ nhỏ, giải bài toán trên các miền đã chia rồi tổng hợp kết quả lại .

Chẳng hạn như thuật toán Quicksort.

b) Phương pháp quay lui (BackTracking method).

Ý tưởng là: Tìm kiếm theo ưu tiên. Đối với mỗi bước thuật toán, ưu tiên theo độ rộng hay chiều sâu để tìm kiếm.

Chẳng hạn thuật toán giải bài toán 8 hậu.

c) Phương pháp tham lam (Greedy Method).

Ý tưởng là : Xác định trật tự xử lý để có lợi nhất, Sắp xếp dữ liệu theo trật tự đó, rồi xử lý dữ liệu theo trật tự đã nêu. Công sức bỏ ra là tìm ra trật tự đó.

Chẳng hạn thuật toán tìm cây bao trùm nhỏ nhất (Shortest spanning Trees).

d) Phương pháp Quy hoạch động (Dynamic Programming method).

Phương pháp quy hoạch động dựa vào một nguyên lý, gọi là nguyên lý tối ưu của Bellman :

- Nếu lời giải của bài toán là tối ưu thì lời giải của các bài toán con cũng tối ưu. Phương pháp này tổ chức tìm kiếm lời giải theo kiểu từ dưới lên. Xuất phát từ các bài toán con nhỏ và đơn giản nhất, tổ hợp các lời giải của chúng để có lời giải của bài toán con lớn hơn...và cứ như thế cuối cùng được lời giải của bài toán ban đầu.

Chẳng hạn thuật toán “chiếc túi xách” (Knapsack).

11.1.6 Các bài tập

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 a)

- *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

11.1.6.1 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

Figure 11.2

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

- Cài đặt thuật toán:
 Input : $a[l..r]$, ($l \leq r$)
 Output: $\text{Min} = \text{Min}(a[l]..a[r])$
 $\text{Max} = \text{Max}(a[l]..a[r])$

```

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;
    }
}

```

Figure 11.3

11.1.6.2 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 a_i và a_j nếu hai phía chưa vượt qua nhau.

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

Kết quả phân hoạch dãy thành 3 phần :

+ $a_k \leq x$ với $k = 1, \dots, j$ (Dãy con thấp);

+ $a_m \geq x$ với $m = i, \dots, n$ (Dãy con cao);

+ $a_h = x$ với $h = j+1, \dots, i+1$

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

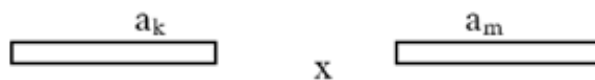


Figure 11.4

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.

11.1.7 BÀI TOÁN THUẬT TOÁN NHÁNH CẬN

11.1.7.1 Bài 1 :

Có n đồ vật, mỗi vật i đặc trưng bởi trọng lượng w_i và giá trị sử dụng v_i , với mọi i thuộc $\{1, \dots, n\}$. Cần chọn các vật này đặt vào một chiếc túi xách có giới hạn trọng lượng m , sao cho tổng giá trị sử dụng các vật được chọn là lớn nhất.

11.1.7.2 Bài 2 :

Cho 3 ký tự A, B, C và n là một số nguyên dương.

Xác định chuỗi tạo ra từ 3 ký tự trên, với chiều dài n , thỏa điều kiện không có 2 chuỗi con liên tiếp nào giống nhau và sao cho số ký tự B là ít nhất.

11.1.7.3 Bài 3:

Tìm lời giải tối ưu cho bài toán:

$$\begin{cases} 10x_1 + 5x_2 + 3x_3 + 6x_4 \rightarrow \text{Max} \\ 5x_1 + 3x_2 + 2x_3 + 4x_4 \leq 8 \\ x_1, x_2, x_3, x_4 \in \mathbb{N} \end{cases}$$

Figure 11.5

11.1.7.4 Bài 4:

Giả sử có n công việc và n thợ. Chi phí trả cho người thợ i để làm công việc j là C_{ij} . Mỗi công việc chỉ do một thợ thực hiện và ngược lại.

Tìm cách thuê các thợ làm việc sao cho tổng chi phí là nhỏ nhất.

Chương 12

Tài liệu tham khảo

12.1 Tài liệu tham khảo¹

- [1] Cấu trúc dữ liệu và thuật toán, Đinh Mạnh Tường, Nhà xuất bản khoa học và kỹ thuật, 2000
- [2] Thiết kế và đánh giá thuật toán, Trần Tuấn Minh, Khoa Toán Tin, Trường Đại học Đà Lạt
- [3] Thiết kế thuật toán, Vũ Đình Hóa, Đỗ Trung Kiên, Đại học sư phạm Hà Nội
- [4] Cẩm nang thuật toán, Robert Sedgewick, Chủ biên dịch: GS. Hoàng Kiếm, NXB KH-KT, 1996.
- [5] Data Structures & Algorithms in Java – SAMS, Robert Lafore, 2001.
- [6] Algorithms, Robert Sedgewick, Addison-Wesley Publishing, 1983
- [7] Garey M.R., Johnson D.S. Computer and intractability.
- [8] NIKLAUS WIRTH , “Algorithms + data structures = Programs”, Prentice-Hall INC, 1976

¹This content is available online at <<http://voer.edu.vn/content/m12979/1.1/>>.

Tham gia đóng góp

Tài liệu: *Thiết kế và đánh giá thuật toán*

Biên soạn bởi: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://voer.edu.vn/content/col10114/1.1/>

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: "Thiết kế đánh giá giải thuật:Lời nói đầu"

Được sử dụng như là: "Lời nói đầu"

Tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://voer.edu.vn/content/m12773/1.1/>

Trang: 1

Bản quyền: Khoa CNTT ĐHSP KT Hưng Yên

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: "Giới thiệu môn học, phương pháp học"

Tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://voer.edu.vn/content/m12765/1.2/>

Trang: 3

Bản quyền: Khoa CNTT ĐHSP KT Hưng Yên

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: "Khái niệm thuật toán:"

Tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://voer.edu.vn/content/m12769/1.1/>

Trang: 3-4

Bản quyền: Khoa CNTT ĐHSP KT Hưng Yên

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: "Các vấn đề liên quan đến thuật toán"

Tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://voer.edu.vn/content/m12666/1.1/>

Trang: 4-5

Bản quyền: Khoa CNTT ĐHSP KT Hưng Yên

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: "Phương pháp liệt kê từng bước"

Tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://voer.edu.vn/content/m12653/1.1/>

Trang: 5-6

Bản quyền: Khoa CNTT ĐHSP KT Hưng Yên

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: "Phương pháp sơ đồ"

Tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://voer.edu.vn/content/m12772/1.1/>

Trang: 6-8

Bản quyền: Khoa CNTT ĐHSP KT Hưng Yên

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: "Mã giả (pseudocode)"

Tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://voer.edu.vn/content/m12660/1.1/>

Trang: 8

Bản quyền: Khoa CNTT ĐHSP KT Hưng Yên

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: "Phân tích thuật toán"

Tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://voer.edu.vn/content/m12783/1.1/>

Trang: 9-10

Bản quyền: Khoa CNTT ĐHSP KT Hưng Yên

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: " $O(f(x))$ và đánh giá thời gian thực hiện thuật toán."

Tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://voer.edu.vn/content/m12785/1.1/>

Trang: 10-11

Bản quyền: Khoa CNTT ĐHSP KT Hưng Yên

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: "Các qui tắc để đánh giá thời gian thực hiện thuật toán"

Tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://voer.edu.vn/content/m12789/1.1/>

Trang: 11-12

Bản quyền: Khoa CNTT ĐHSP KT Hưng Yên

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: "Đánh giá thủ tục (hoặc hàm) đệ qui."

Tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://voer.edu.vn/content/m12795/1.1/>

Trang: 12-15

Bản quyền: Khoa CNTT ĐHSP KT Hưng Yên

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: "Một số phương pháp thiết kế"

Tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://voer.edu.vn/content/m12799/1.1/>

Trang: 15-16

Bản quyền: Khoa CNTT ĐHSP KT Hưng Yên

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: "Cơ bản về thuật toán chia để trị"

Tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://voer.edu.vn/content/m12811/1.1/>

Trang: 17

Bản quyền: Khoa CNTT ĐHSP KT Hưng Yên

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: "Sơ đồ chung của thuật toán"

Tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://voer.edu.vn/content/m12817/1.1/>

Trang: 17-18

Bản quyền: Khoa CNTT ĐHSP KT Hưng Yên

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: "Tìm kiếm nhị phân"

Tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://voer.edu.vn/content/m12836/1.1/>

Trang: 18-19

Bản quyền: Khoa CNTT ĐHSP KT Hưng Yên

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: "Bài toán Min_Max"

Tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://voer.edu.vn/content/m12830/1.1/>

Trang: 20-21

Bản quyền: Khoa CNTT ĐHSP KT Hưng Yên

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: "Thuật toán nhân 2 ma trận"

Tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://voer.edu.vn/content/m12870/1.1/>

Trang: 23-25

Bản quyền: Khoa CNTT ĐHSP KT Hưng Yên

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: "Thuật toán sắp xếp"

Tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://voer.edu.vn/content/m12855/1.1/>

Trang: 25-32

Bản quyền: Khoa CNTT ĐHSP KT Hưng Yên

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: "Bài toán hoán đổi"

Tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://voer.edu.vn/content/m12845/1.1/>

Trang: 32-33

Bản quyền: Khoa CNTT ĐHSP KT Hưng Yên

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: "Sơ đồ chung của thuật toán quay lui"

Tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://voer.edu.vn/content/m12879/1.1/>

Trang: 35-37

Bản quyền: Khoa CNTT ĐHSP KT Hưng Yên

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: "Bài toán ngựa đi tuần"

Tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://voer.edu.vn/content/m12885/1.1/>

Trang: 37-40

Bản quyền: Khoa CNTT ĐHSP KT Hưng Yên

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: "Bài toán 8 quân hậu"

Tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://voer.edu.vn/content/m12876/1.1/>

Trang: 40-42

Bản quyền: Khoa CNTT ĐHSP KT Hưng Yên

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: "Bài toán tìm kiếm đường đi trên đồ thị"

Tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://voer.edu.vn/content/m12893/1.1/>

Trang: 42-43

Bản quyền: Khoa CNTT ĐHSP KT Hưng Yên

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: "Một số bài toán khác"

Tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://voer.edu.vn/content/m12899/1.1/>

Trang: 43-46

Bản quyền: Khoa CNTT ĐHSP KT Hưng Yên

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: "Sơ đồ chung của thuật toán"

Tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://voer.edu.vn/content/m12922/1.1/>

Trang: 47-48

Bản quyền: Khoa CNTT ĐHSP KT Hưng Yên

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: "Bài toán người du lịch"

Tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://voer.edu.vn/content/m12918/1.1/>

Trang: 48-51

Bản quyền: Khoa CNTT ĐHSP KT Hưng Yên

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: "Bài toán cái túi xách"

Tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://voer.edu.vn/content/m12931/1.1/>

Trang: 51-55

Bản quyền: Khoa CNTT ĐHSP KT Hưng Yên

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: "Thuật toán Tham Lam"

Tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://voer.edu.vn/content/m12935/1.1/>

Trang: 57-58

Bản quyền: Khoa CNTT ĐHSP KT Hưng Yên

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: "Bài toán người du lịch"

Tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://voer.edu.vn/content/m12943/1.1/>

Trang: 58-62

Bản quyền: Khoa CNTT ĐHSP KT Hưng Yên

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: "Thuật toán Dijkstra – Tìm đường đi ngắn nhất trong đồ thị có trọng số"

Tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://voer.edu.vn/content/m12950/1.1/>

Trang: 62-65

Bản quyền: Khoa CNTT ĐHSP KT Hưng Yên

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: "Sơ đồ chung của thuật toán"

Tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://voer.edu.vn/content/m12952/1.1/>

Trang: 67-68

Bản quyền: Khoa CNTT ĐHSP KT Hưng Yên

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: "Bài toán thực hiện dãy phép nhân ma trận"

Tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://voer.edu.vn/content/m12957/1.1/>

Trang: 68-72

Bản quyền: Khoa CNTT ĐHSP KT Hưng Yên

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: "Tập độc lớn nhất trên cây"

Tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://voer.edu.vn/content/m12965/1.1/>

Trang: 73-75

Bản quyền: Khoa CNTT ĐHSP KT Hưng Yên

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: "Bài toán dãy con lớn nhất"

Tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://voer.edu.vn/content/m12960/1.1/>

Trang: 75

Bản quyền: Khoa CNTT ĐHSP KT Hưng Yên

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: "Bài toán dãy con chung dài nhất"

Tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://voer.edu.vn/content/m12969/1.1/>

Trang: 76-77

Bản quyền: Khoa CNTT ĐHSP KT Hưng Yên

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: "Bài tập tổng kết"

Tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://voer.edu.vn/content/m12445/1.1/>

Trang: 79-85

Bản quyền: Khoa CNTT ĐHSP KT Hưng Yên

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Module: "Tài liệu tham khảo"

Tác giả: Khoa CNTT ĐHSP KT Hưng Yên

URL: <http://voer.edu.vn/content/m12979/1.1/>

Trang: 87

Bản quyền: Khoa CNTT ĐHSP KT Hưng Yên

Giấy phép: <http://creativecommons.org/licenses/by/3.0/>

Thiết kế và đánh giá thuật toán

Bài giảng về Thiết kế và đánh giá thuật toán

Học liệu Mo Vietnam - Vietnam Open Educational Resources

Học liệu mở Việt Nam là hỗ trợ việc quản lý, tạo, lưu trữ tài liệu giáo dục hiệu quả.