**DIVIDE AND CONQUER**

**KỸ THUẬT THIẾT KẾ THUẬT TOÁN CHIA ĐỂ TRỊ**

***“****Chỉ có hai cách để lãnh đạo: hoặc bạn chia rẽ và chinh phục, hoặc bạn xây dựng và đoàn kết.”*

**Giới thiệu**:

Hãy giả sử, bạn là một vị vua của một vương quốc làm thế nào để bạn có thể cai trị được một vương quốc của chính mình? Đúng rồi, bạn sẽ cho một số người – các vị quan (đây gọi là chia sẽ quyền lực) để đảm nhiệm trọng trách để cai quản một khu vực cụ thể, làm như vậy sẽ giúp bạn dễ dàng hơn trong việc quản lý chất lượng sống của người dân và bảo đảm trật tự xã hội.

Ví dụ cụ thể, bạn muốn biết dân số của đất nước mình là bao nhiêu người, bạn không thể chỉ tới từng nhà và đếm số lượng người, làm như vậy thật tốn công sức và chẳng biết khi nào mới xong. Thay vào đó, nhờ chia sẽ quyền lực như trên, bạn giao công việc cho các vị tỉnh trưởng ở từng khu vực, các tỉnh trưởng sẽ nhờ huyện trưởng, lý trưởng để đếm số dân cụ thể mà mình quản lý rồi tổng hợp lại sẽ cho một con số chính xác và nhanh chóng.

Trong khoa học máy tính có một thuật toán được phát triển dựa trên tư tưởng như trên được gọi với cái tên rất thú vị đó là Chia để trị (Divide and conquer). Hãy cùng tìm hiểu và hy vọng qua bài học ngày hôm nay bạn sẽ tự thiết kế cho mình một thuật toán có tư tưởng như vậy nhé.

**Khái niệm:**

Chia để trị là 1 giải thuật áp dụng cho các bài toán có thể giải quyết bằng cách **chia nhỏ ra thành các bài toán con (có cùng loại)**. Tìm lời **giải của các bài toán con** và **tổng hợp lại** thành lời giải cho bài toán của chúng ta.

Mô hình chia để trị thường được sử dụng để **tìm ra giải pháp tối ưu** cho một vấn đề.

**Lịch sử:**

Các ví dụ **ban đầu của các thuật toán này chủ yếu được giảm để trị (decrease and conquer)** - vấn đề ban đầu **được chia nhỏ liên tục thành các vấn đề con đơn lẻ** và thực sự có thể được giải theo cách lặp đi lặp lại.

**TCN** – Ý tưởng sử dụng danh sách các **mục để sắp xếp đã có từ năm 200 TCN ở Babylon**

– **Euclide** sử dụng ý tưởng chia để trị để tính **ước số chung lớn nhất của hai số nguyên** bằng cách giảm các số thành các bài toán con tương đương nhỏ hơn và nhỏ hơn.

**1805** – Nhà toán học **Gauss** đưa ra ví dụ với nhiều bài toán con mà ngày nay được gọi là **thuật toán biến đổi Fourier nhanh** (FFT), mặc dù ông đã không phân tích số lượng hoạt động của nó.

**1946** – **John Mauchly mô tả rõ ràng về thuật toán** đã được in trên báo.

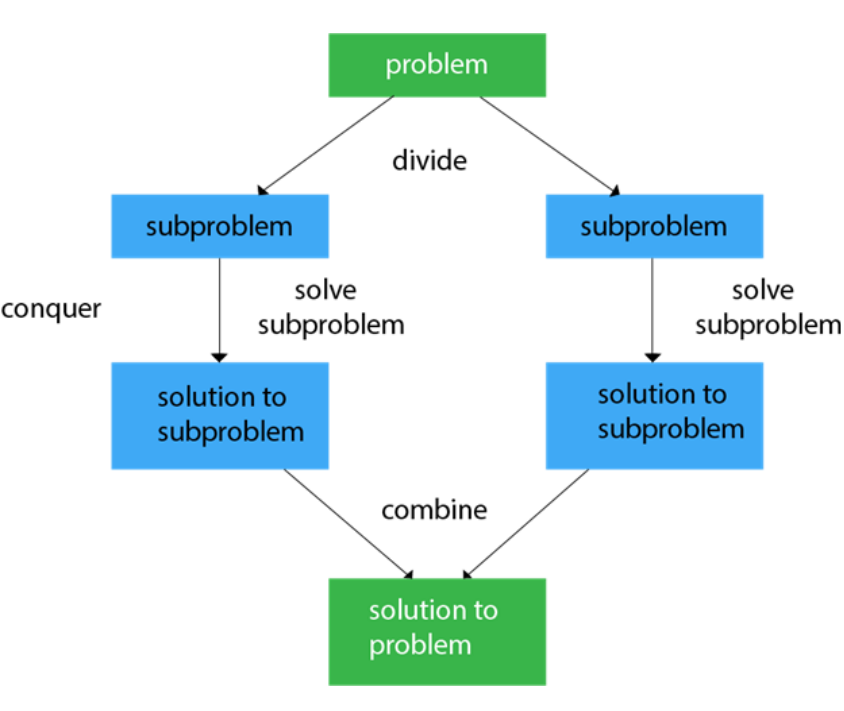
1960 - **Anatolii A. Karatsuba** phát minh, có thể **nhân hai số có n chữ số** trong O(nlog23).

**Các thuật toán chia để trị hoạt động theo những bước sau đây**:

**1**. **Chia (Divide)**: Chia bài toán đã cho thành các bài toán con cùng loại. Bước này liên quan đến việc chia vấn đề thành các vấn đề con nhỏ hơn. **Các vấn đề con nên đại diện cho một phần của vấn đề ban đầu**. Bước này thường sử dụng phương pháp **đệ quy để chia bài toán cho đến khi không còn bài toán con nào** có thể chia tiếp được. Ở giai đoạn này, **các bài toán con trở thành nguyên tử về bản chất nhưng vẫn đại diện cho một số phần của bài toán thực tế**.

**2**. **Trị (Conquer)**: **Đệ quy giải các bài toán con này**. Bước này nhận được rất nhiều bài toán con nhỏ hơn cần giải quyết. Nói chung, **ở cấp độ này, các vấn đề được coi là 'tự giải quyết'.**

**3**. **Kết hợp (Combine)**: Kết hợp các câu trả lời một cách thích hợp**. Khi các bài toán con nhỏ hơn được giải quyết, giai đoạn này kết hợp chúng một cách đệ quy cho đến khi chúng hình thành một giải pháp cho bài toán ban đầu**. Cách tiếp cận theo thuật toán này hoạt động theo cách đệ quy và các bước trị & hợp nhất hoạt động gần đến mức chúng xuất hiện như một.



**Mô tả**: ***Chia một vấn đề thành hai bài toán con nhỏ hơn, tiếp đến tổng hợp các giải quyết của các bài toán con để giải quyết vấn đề ban đầu.***

**Ví dụ:**

Tính tổng của n số a0, a1,…, an-1 ( với n > 1 )

Giải pháp: Chúng ta có thể chia vấn đề thành hai bài toán con có cùng một vấn đề:

1. Tính tổng của nửa đầu của dãy số từ a0 đến an/2-1

2. Tính tổng của nửa đầu của dãy số từ an/2 đến an-1

Khi mỗi tổng trong hai bài toán con này được tính bằng cách áp dụng một phương pháp đệ quy, chúng ta có thể cộng các giá trị của chúng để có được tổng được đề cập:

a0 + … + an-1 = (a0 + … + an/2-1) + (an/2 + … + an-1)

***Đây có phải là một cách hiệu quả để tính tổng n số? Tại sao nó có thể hiệu quả hơn phép tính tổng dùng kỹ thuật vét cạn?***

**Không nhất thiết** mọi thuật toán chia để trị phải hiệu quả cao hơn thuật toán vét cạn.

Có một sự thật là, đa phần các thuật toán dùng kỹ thuật chia để trị **có thời gian nhỏ hơn đáng kể** so với các thuật toán khác.

Trường hợp điển hình nhất của kỹ thuật chia để trị là một vấn đề có kích thước n được chia thành hai trường hợp có kích thước n/2. Tổng quát hơn, một vấn đề có kích thước n có thể được chia thành b các trường hợp có kích thước n/b với một trong số các vấn đề con được giải quyết.

Cho a và b là hằng số; a>=1 và b > 1. Giả sử rằng kích thước n là lũy thừa của b chúng ta phân tích được thời gian chạy T(n) sau đây:

**T(n) = aT(n/b) + f(n)**

**Trong đó:**

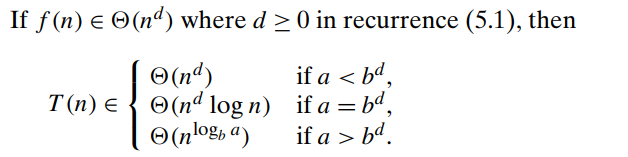
**a** là số bài toán con trong đệ quy.

**n/b** là kích thước của mỗi bài toán.

**f(n)** là một hàm thời gian dành cho việc **chia một vấn đề** có kích thước n thành các vấn đề có kích thước n/b và **kết hợp các giải pháp của chúng**. ( Đối với ví dụ tổng ở trên thì a = b = 2 và f(n) = 1.)

Thứ tự tăng trưởng của T(n) phụ thuộc vào các giá trị của hằng số a, b và thứ tự tăng trưởng của hàm f(n)

**Định lý Thợ:**



Lấy lại ví dụ tính tổng A(n) ở trên thì đầu vào có kích thước n = 2k

**A(n) = 2A(n/2) + 1.**

Vì vậy, trong ví dụ này, a = 2, b = 2 và d = 0; do đó a > bd



**Ứng dụng:**

Mặc dù giải thuật chia để trị không thể áp dụng hay hiệu suất kém hơn các giải thuật đơn giản khác, nhưng nhiều thuật toán hiệu quả đã phát triển dựa trên giải thuật chia để trị này.

Là một kỹ thuật hiệu quả cho nhiều vấn đề như:

**Sắp xếp – Sorting (Merge Sort, QuickSort)**

Vd: Mergesort độ hiệu quả thời gian là theta(nlogn) trong mọi trường hợp, với số lượng do phép tính cơ bản chính (phép so sánh) rất gần với tối tiểu lý thuyết. Hạn chế chính của nó là yêu cầu bộ nhớ đáng kể.

**Nhân số lớn trong Mật mã học – Multiplying large numbers**

­­Ứng dụng của giải thuật chia để trị **được áp dụng trong mật mã học** khi mà độ dài của số nguyên có thể lên hơn 100 chữ số - điều này là quá dài đối với một từ duy nhất đối với máy tính ngày nay. Lời giải thuật này hiệu suất dựa vào hệ thống máy tính và chương trình thực hiện thuật toán. Trên một số máy, thuật toán chia để trị đã được báo cáo là hoạt động **tốt hơn phương pháp thông thường trên các số chỉ dài 8 chữ số thập phân và chạy nhanh hơn gấp đôi với các số dài hơn 300 chữ số thập phân** - lĩnh vực đặc biệt quan trọng đối với mật mã học hiện đại.

Làm giảm thời gian để nhân 2 số có độ dài n từ O(n2) xuống còn n1.585

**Nhân ma trận - Strassen’s algorithm**

Ứng dụng trong việc nhân ma trận. Cái nhìn sâu sắc chính của thuật toán nằm ở việc khám phá ra rằng chúng ta có thể tìm thấy tích C của hai ma trận 2 × 2 A và B chỉ với bảy phép nhân trái ngược với tám yêu cầu của thuật toán brute-force. Tầm quan trọng của nó bắt nguồn từ ưu thế tiệm cận của nó khi thứ tự ma trận n đi đến vô cùng.

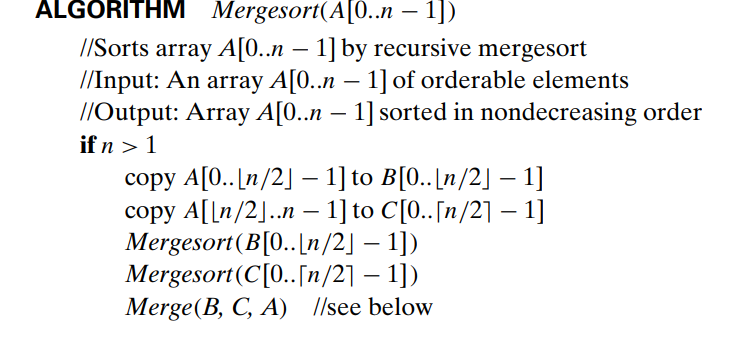
Giảm từ O(n3) xuống còn O(n2.807). Một mặt, chúng ngày càng tiến gần hơn đến cận dưới lý thuyết tốt nhất được biết đến với phép nhân ma trận, đó là phép nhân n2, mặc dù hiện thuật toán vẫn còn chưa được giải quyết.­

**Hình học toán học – finding the closest pair of points, Convex Hull**

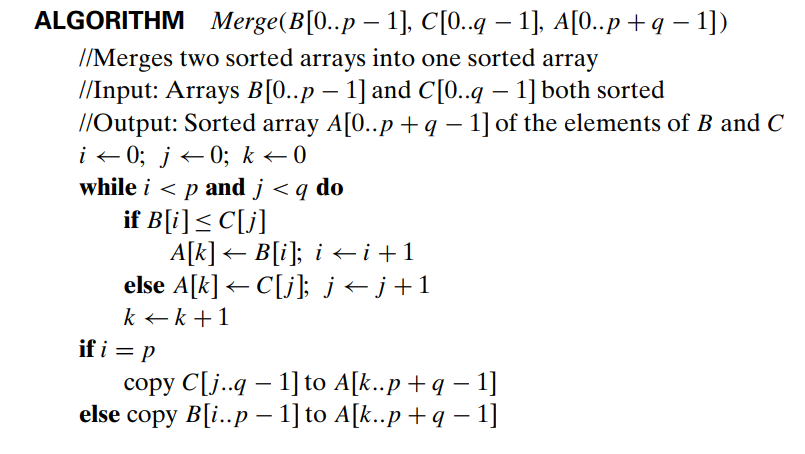
Giải thuật chia để trị có thể được áp dụng thành công cho hai vấn đề quan trọng của hình học tính toán: bài toán cặp gần nhất (**finding the closest pair of points**) và bài toán thân lồi (**Convex Hull**)

**THUẬT TOÁN MERGESORT.**

Mergesort là một ví dụ hoàn hảo về việc áp dụng thành công kỹ thuật chia để trị. Nó sắp xếp một mảng A[0…n-1] bằng cách chia thành hai nửa A[0 … n/2 – 1] và A[n/2 … n-1] sắp xếp đệ quy và sau đó hợp nhất hai mảng đã được sắp xếp thành một mảng được sắp xếp duy nhất



Việc hợp nhất hai mảng được sắp xếp có thể được thực hiện như sau. Hai con trỏ (chỉ số mảng) được khởi tạo để trỏ đến các phần tử đầu tiên của mảng được hợp nhất. Các phần tử được chỉ ra được so sánh và các phần tử nhỏ hơn được thêm vào một mảng mới đang được xây dựng; Sau đó, chỉ số của phần tử nhỏ hơn được tăng lên để chỉ đến người kế nhiệm trực tiếp của nó trong mảng mà nó được sao chép từ đó. Thao tác này được lặp lại cho đến khi một trong hai mảng đã cho hết, và sau đó các phần tử còn lại của mảng khác được sao chép vào cuối mảng mới.



Giả sử n là lũy thừa của 2, số phép so sánh là C(n) là:

**C(n) = 2C(n/2) + Cmerge(n) . Với n > 1, C(1) = 0.**

Chúng ta hãy phân tích Cmerge(n), số lượng so sánh chính được thực hiện trong giai đoạn hợp nhất. Ở mỗi bước có 1 bước so sánh được thực hiện, sau đó tổng số phần tử trong hai mảng vẫn cần được xử lý giảm 1. Trường hợp xấu nhất, không có mảng nào rỗng trước (hết phần tử để so sánh). Vì vậy, trong trường hợp tệ nhất, Cmerge(n) = n – 1, và chúng ta có:

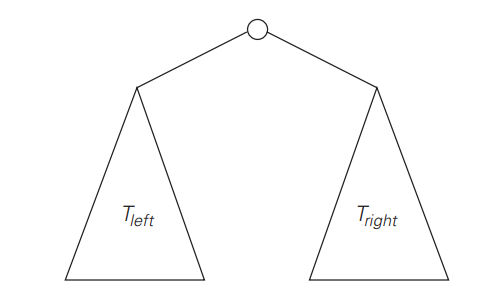
**Cworst(n) = 2Cworst(n/2) + n – 1 . Với n > 1, Cworst(1) = 0**

Do đó theo định lý thợ: Cworst(n) ∈ theta(nlogn). Trường hợp xấu nhất cho n = 2k

**Cworst(n) = nlog2n – n + 1**

**DUYỆT CÂY NHỊ PHÂN**

Trong phần này, chúng ta thấy làm thế nào kỹ thuật chia để trị có thể được áp dụng cho cây nhị phân. **Một cây nhị phân T được định nghĩa là một tập hợp hữu hạn các nút trống hoặc bao gồm một gốc và hai cây nhị phân rời rạc TL và TR** được gọi tương ứng là cây con bên trái và bên phải của gốc



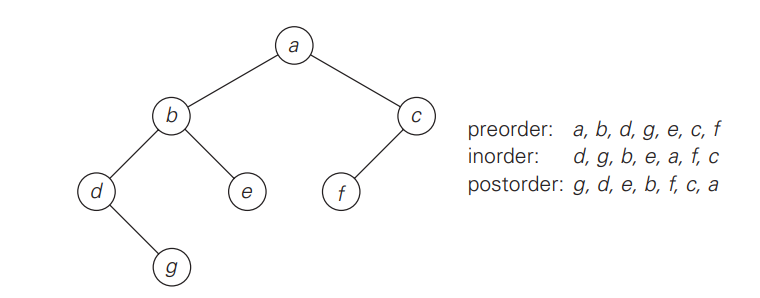
Vì bản thân định nghĩa chia một cây nhị phân thành hai cấu trúc nhỏ hơn cùng loại, cây con bên trái và cây con bên phải, nhiều vấn đề về cây nhị phân có thể được giải quyết bằng cách áp dụng kỹ thuật chia để trị.

Các thuật toán chia để trị quan trọng nhất cho cây nhị phân là ba phép duyệt cổ điển: **preorder, inorder và postorder**. Tất cả ba phép duyệt ghé thăm các nút của cây nhị phân đệ quy, tức là, bằng cách truy cập gốc của cây và bên trái của nó và cây con bên phải. Chúng chỉ khác nhau bởi thời gian truy cập của root:

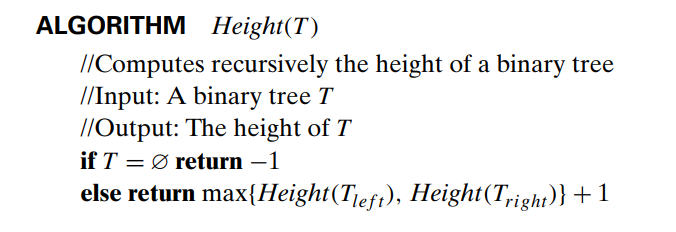
**Trong phép duyệt preorder**, gốc được truy cập trước khi các cây con bên trái và bên phải được truy cập (theo thứ tự đó).

**Trong phép duyệt inorder**, gốc được truy cập sau khi truy cập vào cây con bên trái của nó nhưng trước khi truy cập vào cây con bên phải.

**Trong phép duyệt postorder**, gốc được truy cập sau khi truy cập các cây con bên trái và bên phải (theo thứ tự đó).



Ví dụ, chúng ta hãy xem xét một thuật toán đệ quy để **tính chiều cao của cây nhị phân**. Hãy nhớ lại rằng chiều cao được định nghĩa là **chiều dài của con đường dài nhất từ gốc đến lá**. Do đó, nó có thể được tính là chiều cao tối đa của chiều cao bên trái của gốc và cây con bên phải cộng với 1. (Chúng ta phải thêm 1 vào cho cấp độ bổ sung của gốc.) Cũng lưu ý rằng thật thuận tiện để xác định **chiều cao của cây trống là −1**. Như vậy, chúng ta có thuật toán đệ quy sau:

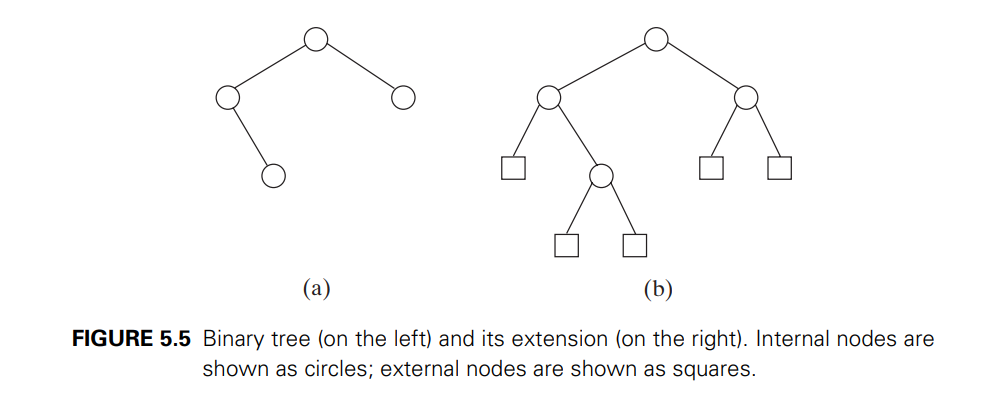


Kích thước đầu vào của vấn đề bằng số nút n (T) trong một cây nhị phân T nhất định. Rõ ràng, số lượng so sánh được thực hiện để tính tối đa hai số và số phép cộng A (n (T)) được thực hiện bởi thuật toán là như nhau. Chúng ta có mối quan hệ lặp lại sau đây cho A(n(T)):

**A(n(T)) = A(n(Tleft)) + A(n(Tright)) + 1. Với n(T) > 0, A(0) = 0.**

Chúng ta hãy lưu ý rằng phép cộng không phải là hoạt động được thực hiện thường xuyên nhất của thuật toán này mà là **phép kiểm tra** và điều này rất điển hình cho các thuật toán cây nhị phân - rằng cây không trống. Ví dụ, đối với cây trống, phép so sánh T = ∅ được thực hiện một lần nhưng không có phép cộng và đối với cây một nút, số so sánh và số cộng lần lượt là 3 và 1.

Nó giúp phân tích các thuật toán cây để vẽ phần mở rộng của cây bằng cách thay thế các cây con trống bằng các nút đặc biệt. Các nút phụ (được hiển thị bằng các ô vuông nhỏ trong Hình 5.5) được gọi là bên ngoài; Các nút ban đầu (được hiển thị bởi ít vòng tròn) được gọi là nội bộ. Theo định nghĩa, phần mở rộng của cây nhị phân trống là một nút bên ngoài duy nhất.



Thật dễ dàng để thấy rằng thuật toán Chiều cao thực hiện chính xác một bổ sung cho mỗi nút bên trong của cây mở rộng và nó thực hiện một so sánh để kiểm tra xem cây trống cho mọi nút bên trong và bên ngoài. Do đó, để xác định hiệu quả của thuật toán, chúng ta cần biết một cây nhị phân mở rộng với n nút bên trong có thể có bao nhiêu nút bên ngoài. Sau khi kiểm tra Hình 5.5 và một vài các ví dụ tương tự, thật dễ dàng để đưa ra giả thuyết rằng số lượng nút bên ngoài x luôn nhiều hơn 1 so với số nút bên trong n:

**x = n + 1.**

Để chứng minh sự bình đẳng này, hãy xem xét tổng số nút, cả bên trong và bên ngoài. Vì mọi nút, ngoại trừ gốc, là một trong hai con của một nút trong, chúng ta có phương trình.

**2n + 1 = x + n**

Công thức trên cũng áp dụng cho bất kỳ cây nhị phân đầy đủ không rỗng nào, trong đó, theo định nghĩa, mọi nút đều có 0 hoặc hai con: đối với một cây nhị phân đầy đủ, n và x biểu thị số nút cha và lá tương ứng.

**C(n) = n + x = 2n + 1**

Và số lượng bổ sung là

**A(n) = n**

**Ưu điểm:**

**Giải quyết các bài toán khó:**

Chia để trị là một công cụ mạnh mẽ để giải quyết các vấn đề khó về mặt khái niệm: tất cả những gì nó cần là một cách chia nhỏ vấn đề thành các vấn đề con, giải quyết các trường hợp tầm thường và kết hợp các vấn đề con với vấn đề ban đầu. ***Tương tự, giảm và chinh phục chỉ yêu cầu giảm vấn đề thành một vấn đề nhỏ hơn.*** Ví dụ như bài toán Tháp Hà Nội giúp giảm việc di chuyển một tòa tháp có chiều cao n thành di chuyển một tháp có chiều cao n - 1

**Độ hiệu quả của thuật toán:**

Mô hình chia để trị thường giúp phát hiện ra các thuật toán hiệu quả. Ví dụ, nó là chìa khóa cho phương pháp nhân nhanh của Karatsuba, thuật toán Quicksort và Mergesort, thuật toán Strassen cho phép nhân ma trận và biến đổi Fourier nhanh.

Trong tất cả các ví dụ này, phương pháp D&C đã dẫn đến sự cải thiện về chi phí tiệm cận của giải pháp. Ví dụ: nếu (a) các trường hợp cơ sở có kích thước giới hạn không đổi, công việc chia tách vấn đề và kết hợp các giải pháp từng phần tỷ lệ thuận với kích thước của vấn đề n và (b) có một số bị chặn p của các bài toán con về kích thước ~ n/p ở mỗi giai đoạn, thì chi phí của thuật toán chia để trị sẽ là *O*(nlogpn)

**Song song:**

Các thuật toán chia để trị được điều chỉnh một cách tự nhiên để thực thi trong **các máy đa bộ xử lý**, đặc biệt là các **hệ thống chia sẻ bộ nhớ** nơi mà việc truyền dữ liệu giữa các bộ xử lý không cần phải lên kế hoạch trước vì các vấn đề con riêng biệt **có thể được thực thi trên các bộ xử lý khác nhau**.

**Truy cập bộ nhớ:**

Các thuật toán chia để trị thường có xu hướng sử dụng hiệu quả bộ nhớ cache. Lý do là một khi vấn đề phụ đủ nhỏ, về nguyên tắc, nó và tất cả các vấn đề phụ của nó có thể được ***giải quyết trong bộ đệm mà không cần truy cập vào bộ nhớ chính chậm hơn***. Một thuật toán được thiết kế để khai thác bộ đệm theo cách này được gọi là ***không nhớ bộ đệm***, vì nó không chứa kích thước bộ đệm dưới dạng tham số rõ ràng. Hơn nữa, các thuật toán D&C có thể được thiết kế cho các thuật toán quan trọng (ví dụ: sắp xếp, FFT và phép nhân ma trận) để trở thành các thuật toán bỏ qua bộ đệm tối ưu–chúng sử dụng bộ đệm theo cách có thể là tối ưu, theo nghĩa tiệm cận, bất kể kích thước bộ đệm. Ngược lại, cách tiếp cận truyền thống để khai thác bộ nhớ cache là chặn, như trong tối ưu hóa tổ vòng lặp, trong đó vấn đề được chia rõ ràng thành các phần có kích thước phù hợp—điều này cũng có thể sử dụng bộ nhớ cache một cách tối ưu, nhưng chỉ khi thuật toán được điều chỉnh cho mục đích cụ thể kích thước bộ đệm của một máy cụ thể.

Lợi thế tương tự tồn tại đối với các hệ thống lưu trữ phân cấp khác, chẳng hạn như NUMA hoặc bộ nhớ ảo, cũng như đối với nhiều cấp độ bộ đệm: một khi vấn đề con đủ nhỏ, nó có thể được giải quyết trong một cấp độ nhất định của hệ thống phân cấp mà không cần truy cập các cấp độ cao hơn (chậm hơn).

**Kiểm soát làm tròn số:**

Trong các tính toán với số học làm tròn, ví dụ: với các số dấu phẩy động, thuật toán chia để trị có thể mang lại kết quả chính xác hơn so với phương pháp lặp tương đương bề ngoài. Ví dụ: người ta có thể cộng N số bằng một vòng lặp đơn giản thêm từng mốc thời gian vào một biến duy nhất hoặc bằng thuật toán D&C được gọi là phép tính tổng theo cặp để chia tập dữ liệu thành hai nửa, tính toán đệ quy tổng của mỗi nửa rồi cộng hai tổng. Mặc dù phương pháp thứ hai thực hiện cùng một số lần bổ sung như phương pháp thứ nhất và trả phí cho các lệnh gọi đệ quy, nhưng phương pháp này thường chính xác hơn.

**Nhược điểm:**

**1**. **Việc thực hiện phân chia để trị yêu cầu quản lý bộ nhớ cao.**

Khi triển khai đệ quy các thuật toán D&C, người ta phải đảm bảo rằng có đủ bộ nhớ được phân bổ cho ngăn xếp đệ quy, nếu không, việc thực thi có thể thất bại do tràn ngăn xếp. Ví dụ, thuật toán Quicksort có thể được triển khai sao cho nó không bao giờ yêu cầu nhiều hơn log2n gọi đệ quy lồng nhau để sắp xếp n.

Tràn ngăn xếp có thể khó tránh khi sử dụng các thủ tục đệ quy vì nhiều trình biên dịch cho rằng ngăn xếp đệ quy là một vùng liền kề của bộ nhớ và một số phân bổ một lượng không gian cố định cho nó. Trình biên dịch cũng có thể lưu nhiều thông tin hơn mức cần thiết trong ngăn xếp đệ quy, chẳng hạn như địa chỉ trả về, các tham số không thay đổi và các biến nội bộ của thủ tục. Do đó, có thể giảm nguy cơ tràn ngăn xếp bằng cách giảm thiểu các tham số và biến nội bộ của thủ tục đệ quy hoặc bằng cách sử dụng cấu trúc ngăn xếp rõ ràng.

**2**. **Lựa chọn các trường hợp cơ sở.**

Trong bất kỳ thuật toán đệ quy nào, có sự tự do đáng kể trong việc lựa chọn các trường hợp cơ bản, các bài toán con nhỏ được giải trực tiếp để kết thúc đệ quy.

Việc chọn các trường hợp cơ sở nhỏ nhất hoặc đơn giản nhất có thể sẽ thanh lịch hơn và thường dẫn đến các chương trình đơn giản hơn, bởi vì có ít trường hợp cần xem xét hơn và chúng dễ giải quyết hơn.

Mặt khác, hiệu quả thường được cải thiện nếu dừng đệ quy ở các trường hợp cơ sở tương đối lớn và chúng được giải quyết không đệ quy, dẫn đến thuật toán lai (vừa sử dụng đệ quy đến trường hợp cơ sở lại sử dụng thuật toán không đệ quy ). Chiến lược này tránh được chi phí hoạt động của các cuộc gọi đệ quy ít hoặc không hoạt động và cũng có thể cho phép sử dụng các thuật toán không đệ quy chuyên dụng, đối với các trường hợp cơ sở đó, hiệu quả hơn so với đệ quy rõ ràng.

**Tổng kết**

Như vậy là chúng ta đã cùng tìm hiểu về phương pháp thiết kế giải thuật chia để trị, cùng ứng dụng của nó vào việc giải một số bài toán cụ thể. Nhìn chung tư tưởng của chia để trị là chia bài toán lớn thành nhiều bài toán nhỏ, và giải quyết các bài toán nhỏ đó rồi ghép lời giải lại để được lời giải cho bài toán lớn. Thế nên chia để trị thường có đặc điểm như sau:

Sử dụng các lời gọi đệ quy, để giải quyết các bài toán nhỏ bằng cách chia chúng thành các bài toán nhỏ hơn. Thực ra việc sử dụng đệ quy này là không bắt buộc, ta có thể viết lại bằng vòng lặp để khử đệ quy, tuy nhiên cách viết đệ quy vẫn là cách ngắn gọn và dễ hiểu hơn

Có từ 2 lời gọi đệ quy trở lên, do việc cần chia bài toán lớn thành nhiều bài toán nhỏ hơn (và để giải quyết 1 bài toán nhỏ đó, ta cần 1 lời gọi đệ quy). Thực ra tên gọi "chia để trị" thông thường cũng được áp dụng cho các thuật toán quy bài toán ban đầu **về đúng một bài toán nhỏ hơn**, chẳng hạn như thuật toán **tìm kiếm nhị phân**, dùng cho việc tìm khóa trong một danh sách đã sắp xếp. Tuy nhiên nhiều người cũng cho rằng "chia để trị" **chỉ nên dùng cho trường hợp mỗi bài toán có thể được chia thành hai hay nhiều hơn bài toán nhỏ**, khi đó, cái tên **"giảm để trị"** **(decrease and conquer)** được đề xuất dùng cho trường hợp **bài toán lớn được quy về đúng một bài toán nhỏ hơn**

Các bài toán con thường là độc lập với nhau (giải quyết bài toán con này không phụ thuộc vào giải quyết bài toán con kia)