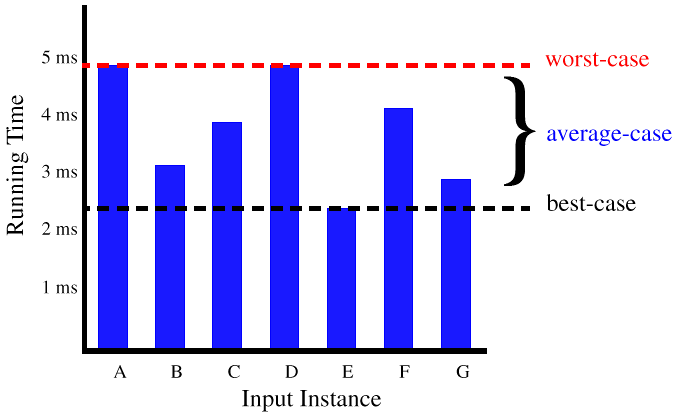
**CHƯƠNG i: CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT**

**Phân tích thời gian thực hiện thuật toán**

**ĐỘ PHỨC TẠP GIẢI THUẬT**

**Giới thiệu**

Hầu hết các bài toán đều có nhiều thuật toán khác nhau để giải quyết chúng. Như vậy, làm thế nào để chọn được sự cài đặt tốt nhất? Đây là một lĩnh vực được phát triển tốt trong nghiên cứu về khoa học máy tính. Chúng ta sẽ thường xuyên có cơ hội tiếp xúc với các kết quả nghiên cứu mô tả các tính năng của các thuật toán cơ bản. Tuy nhiên, việc so sánh các thuật toán rất cần thiết và chắc chắn rằng một vài dòng hướng dẫn tổng quát về phân tích thuật toán sẽ rất hữu dụng.



Khi nói đến hiệu quả của một thuật toán, người ta thường quan tâm đến chi phí cần dùng để thực hiện nó. Chi phí này thể hiện qua việc sử dụng tài nguyên như bộ nhớ, thời gian sử dụng CPU, … Ta có thể đánh giá thuật toán bằng phương pháp thực nghiệm thông qua việc cài đặt thuật toán rồi chọn các bộ dữ liệu thử nghiệm. Thống kê các thông số nhận được khi chạy các dữ liệu này ta sẽ có một đánh giá về thuật toán.

Tuy nhiên, phương pháp thực nghiệm gặp một số nhược điểm sau khiến cho nó khó có khả năng áp dụng trên thực tế:

* Do phải cài đặt bắng một ngôn ngữ lập trình cụ thể nên thuật toán sẽ chịu sự hạn chế của ngữ lập trình này.
* Đồng thời, hiệu quả của thuật toán sẽ bị ảnh hưởng bởi trình độ của người cài đặt.
* Việc chọn được các bộ dữ liệu thử đặc trưng cho tất cả tập các dữ liệu vào của thuật toán là rất khó khăn và tốn nhiều chi phí.
* Các số liệu thu nhận được phụ thuộc nhiều vào phần cứng mà thuật toán được thử nghiệm trên đó. Điều này khiến cho việc so sánh các thuật toán khó khăn nếu chúng được thử nghiệm ở những nơi khác nhau.

Vì những lý do trên, người ta đã tìm kiếm những phương pháp đánh giá thuật toán hình thức hơn, ít phụ thuộc môi trường cũng như phần cứng hơn. Một phương pháp như vậy là phương pháp đánh giá thuật toán theo hướng xầp xỉ tiệm cận qua các khái niệm toán học O-lớn O(), O-nhỏ o()

Thông thường các vấn đề mà chúng ta giải quyết có một "kích thước" tự nhiên (thường là số lượng dữ liệu được xử lý) mà chúng ta sẽ gọi là N. Chúng ta muốn mô tả tài nguyên cần được dùng (thông thường nhất là thời gian cần thiết để giải quyết vấn đề) như một hàm số theo N. Chúng ta quan tâm đến trường hợp trung bình, tức là thời gian cần thiết để xử lý dữ liệu nhập thông thường, và cũng quan tâm đến trường hợp xấu nhất, tương ứng với thời gian cần thiết khi dữ liệu rơi vào trường hợp xấu nhất có thể có.

Việc xác định chi phí trong trường hợp trung bình thường được quan tâm nhiều nhất vì nó đại diện cho đa số trường hợp sử dụng thuật toán. tuy nhiên, việc xác định chi phí trung bình này lại gặp nhiều khó khăn. Vì vậy, trong nhiều trường hợp, người ta xác định chi phí trong trường hợp xấu nhất (chặn trên) thay cho việc xác định chi phí trong trường hợp trung bình. Hơn nữa, trong một số bài toán, việc xác định chi phí trong trường hợp xấu nhất là rất quan trọng. Ví dụ, các bài toán trong hàng không, phẫu thuật, …

**Các bước phân tích thuật toán**

Bước đầu tiên trong việc phân tích một thuật toán là xác định đặc trưng dữ liệu sẽ được dùng làm dữ liệu nhập của thuật toán và quyết định phân tích nào là thích hợp. Về mặt lý tưởng, chúng ta muốn rằng với một phân bố tùy ý được cho của dữ liệu nhập, sẽ có sự phân bố tương ứng về thời gian hoạt động của thuật toán. Chúng ta không thể đạt tới điều lý tưởng nầy cho bất kỳ một thuật toán không tầm thường nào, vì vậy chúng ta chỉ quan tâm đến bao của thống kê về tính năng của thuật toán bằng cách cố gắng chứng minh thời gian chạy luôn luôn nhỏ hơn một "chận trên" bất chấp dữ liệu nhập như thế nào và cố gắng tính được thời gian chạy trung bình cho dữ liệu nhập "ngẫu nhiên".

Bước thứ hai trong phân tích một thuật toán là nhận ra các thao tác trừu tượng của thuật toán để tách biệt sự phân tích với sự cài đặt. Ví dụ, chúng ta tách biệt sự nghiên cứu có bao nhiêu phép so sánh trong một thuật toán sắp xếp khỏi sự xác định cần bao nhiêu micro giây trên một máy tính cụ thể; yếu tố thứ nhất được xác định bởi tính chất của thuật toán, yếu tố thứ hai lại được xác định bởi tính chất của máy tính. Sự tách biệt này cho phép chúng ta so sánh các thuật toán một cách độc lập với sự cài đặt cụ thể hay độc lập với một máy tính cụ thể.

Bước thứ ba trong quá trình phân tích thuật toán là sự 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. Chúng ta sẽ không gặp khó khăn khi tìm một chặn trên cho thời gian chạy chương trình, vấn đề ở chỗ là phải tìm ra chận trên tốt nhất, tức là thời gian chạy chương trình khi gặp dữ liệu nhập của trường hợp xấu nhất. Trường hợp trung bình thông thường đòi hỏi một phân tích toán học tinh vi hơn trường hợp xấu nhất. Mỗi khi đã hoàn thành một quá trình phân tích thuật toán dựa vào các đại lượng cơ bản, nếu thời gian kết hợp với mỗi đại lượng được xác định rõ thì ta sẽ có các biểu thức để tính thời gian chạy.

Nói chung, tính năng của một thuật toán thường có thể được phân tích ở một mức độ vô cùng chính xác, chỉ bị giới hạn bởi tính năng không chắc chắn của máy tính hay bởi sự khó khăn trong việc xác định các tính chất toán học của một vài đại lượng trừu tượng. Tuy nhiên, thay vì phân tích một cách chi tiết chúng ta thường thích ước lượng để tránh sa vào chi tiết.

Cách đánh giá thời gian thực hiện giải thuật độc lập với máy tính và các yếu tố liên quan tới máy như vậy sẽ dẫn đến khái niệm về “ cấp độ lớn của thời gian thực hiện giải thuật” hay nói cách khác là “*độ phức tạp tính toán của giải thuật*”

Nếu thời gian thực hiện một giải thuật là T(n) = cn2 (c = const) thì ta nói độ phức tạp tính toán của giải thuật này có cấp là n2.

Kí hiệu : T(n) = O(n2) (kí hiệu chữ O lớn).

Định nghĩa:

Một hàm f(n) được xác định là O(g(n)) hay f(n) = O(g(n)) và được gọi là có cấp g(n) nếu tồn tại các hằng số c và n0 sao cho :

f(n) ≤ cg(n) khi n ≥ n0

nghĩa là f(n) bị chặn trên bởi một hằng số nhân với g(n), với mọi giá trị của n từ một điểm nào đó.

**Sự  phân lớp các thuật toán**

Như đã được chú ý trong ở trên, hầu hết các thuật toán đều có một tham số chính là N, thông thường đó là số lượng các phần tử dữ liệu được xử lý mà ảnh hưởng rất nhiều tới thời gian chạy. Tham số N có thể là bậc của một đa thức, kích thước của một tập tin được sắp xếp hay tìm kiếm, số nút trong một đồ thị .v.v... Hầu hết tất cả các thuật toán trong giáo trình này có thời gian chạy tiệm cận tới một trong các hàm sau:

Hằng số: Hầu hết các chỉ thị của các chương trình đều được thực hiện một lần hay nhiều nhất chỉ một vài lần. Nếu tất cả các chỉ thị của cùng một chương trình có tính chất nầy thì chúng ta sẽ nói rằng thời gian chạy của nó là hằng số. Điều nầy hiển nhiên là hoàn cảnh phấn đấu để đạt được trong việc thiết kế thuật toán.

logN: Khi thời gian chạy của chương trình là logarit tức là thời gian chạy chương trình tiến chậm khi N lớn dần. Thời gian chạy thuộc loại nầy xuất hiện trong các chương trình mà giải một bài toán lớn bằng cách chuyển nó thành một bài toán nhỏ hơn, bằng cách cắt bỏ kích thước bớt một hằng số nào đó. Với mục đích của chúng ta, thời gian chạy có được xem như nhỏ hơn một hằng số "lớn". Cơ số của logarit làm thay đổi hằng số đó nhưng không nhiều: khi N là một ngàn thì logN là 3 nếu cơ số là 10, là 10 nếu cơ số là 2; khi N là một triệu, logN được nhân gấp đôi. Bất cứ khi nào N được nhân đôi, logN tăng lên thêm một hằng số, nhưng logN không bị nhân gấp đôi khi N tăng tới N2.

N: Khi thời gian chạy của một chương trình là tuyến tính, nói chung đây trường hợp mà một số lượng nhỏ các xử lý được làm cho mỗi phần tử dữ liệu nhập. Khi N là một triệu thì thời gian chạy cũng cỡ như vậy. Khi N được nhân gấp đôi thì thời gian chạy cũng được nhân gấp đôi. Đây là tình huống tối ưu cho một thuật toán mà phải xử lý N dữ liệu nhập (hay sản sinh ra N dữ liệu xuất).

NlogN: Đây là thời gian chạy tăng dần lên cho các thuật toán mà giải một bài toán bằng cách tách nó thành các bài toán con nhỏ hơn, kế đến giải quyết chúng một cách độc lập và sau đó tổ hợp các lời giải. Bởi vì thiếu một tính từ tốt hơn (có lẻ là "tuyến tính logarit"?), chúng ta nói rằng thời gian chạy của thuật toán như thế là "NlogN". Khi N là một triệu, NlogN có lẽ khoảng hai mươi triệu. Khi N được nhân gấp đôi, thời gian chạy bị nhân lên nhiều hơn gấp đôi (nhưng không nhiều lắm).

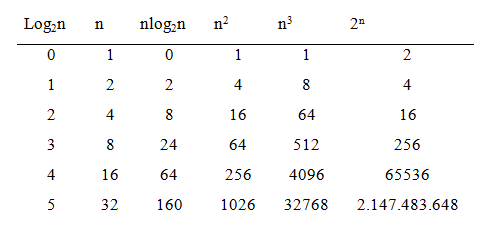
N2: Khi thời gian chạy của một thuật toán là bậc hai, trường hợp nầy chỉ có ý nghĩa thực tế cho các bài toán tương đối nhỏ. Thời gian bình phương thường tăng dần lên trong các thuật toán mà xử lý tất cả các cặp phần tử dữ liệu (có thể là hai vòng lặp lồng nhau). Khi N là một ngàn thì thời gian chạy là một triệu. Khi N được nhân đôi thì thời gian chạy tăng lên gấp bốn lần.

N3:Tương tự, một thuật toán mà xử lý các bộ ba của các phần tử dữ liệu (có lẻ là ba vòng lặp lồng nhau) có thời gian chạy bậc ba và cũng chỉ có ý nghĩa thực tế trong các bài toán nhỏ. Khi N là một trăm thì thời gian chạy là một triệu. Khi N được nhân đôi, thời gian chạy tăng lên gấp tám lần.

2N: Một số ít thuật toán có thời gian chạy lũy thừa lại thích hợp trong một số trường hợp thực tế, mặc dù các thuật toán như thế là "sự ép buộc thô bạo" để giải các bài toán. Khi N là hai mươi thì thời gian chạy là một triệu. Khi N gấp đôi thì thời gian chạy được nâng lên lũy thừa hai!

Thời gian chạy của một chương trình cụ thể đôi khi là một hệ số hằng nhân với các số hạng nói trên ("số hạng dẫn đầu") cộng thêm một số hạng nhỏ hơn. Giá trị của hệ số hằng và các số hạng phụ thuộc vào kết quả của sự phân tích và các chi tiết cài đặt. Hệ số của số hạng dẫn đầu liên quan tới số chỉ thị bên trong vòng lặp: ở một tầng tùy ý của thiết kê thuật toán thì phải cẩn thận giới hạn số chỉ thị như thế. Với N lớn thì các số hạng dẫn đầu đóng vai trò chủ chốt; với N nhỏ thì các số hạng cùng đóng góp vào và sự so sánh các thuật toán sẽ khó khăn hơn. Trong hầu hết các trường hợp, chúng ta sẽ gặp các chương trình có thời gian chạy là "tuyến tính", "NlogN", "bậc ba", ... với hiểu ngầm là các phân tích hay nghiên cứu thực tế phải được làm trong trường hợp mà tính hiệu quả là rất quan trọng.

Sau đây là bảng giá trị của một số hàm đó:



**CÁC QUY TẮC XÁC ĐỊNH ĐỘ PHỨC TẠP GIẢI THUẬT**

+ *Qui tắc cộng:* Giả sử T1(n) và T2(n) là thời gian thực hiện của hai đoạn chương trình P1 và P2 mà :

T1(n) = O(f(n)); T2 = (O(g(n))

thì thời gian thực hiện P1 rồi P2 tiếp theo sẽ là :

T1(n) + T2(n) = O(max (f(n), g(n))

Ví dụ : Trong một chương trình có 3 bước thực hiện mà thời gian thực hiện từng bước lần lượt là O(n2), O(n3) và O(nlog2n) thì thời gian thực hiện 2 bước đầu là O(max(n2, n3)) = O(n3). Thời gian thực hiện chương trình sẽ là O(max(n3, nlog2n)) = O(n3)

Chú ý : Nếu g(n) ≤ f(n) với mọi n ≥ n0 thì O(f(n)+g(n)) cũng là O(f(n)).

VD : O(n4 + n2) = O(n4); O(n + log2n) = O(n).

+ *Qui tắc nhân*: Nếu T1(n) và T2(n) là thời gian thực hiện của 2 đoạn chương trình P1 và P2 trong đó (T1(n) = O(f(n)); T2 = (O(g(n))); thì thời gian thực hiện P1 và P2 lồng nhau là:

T1(n)T2(n) = O(f(n)g(n));

Ví dụ: Câu lệnh For( i = 1 ,i < n , i++) x = x + 1;

có thời gian thực hiện O(n.1) = O(n)

Câu lệnh For( i = 1, i <= n , i++)

For( j = 1 , j <= n , j++)

x = x + 1;

Có thời gian thực hiện được đánh giá là O(n.n) = O(n2)

Chú ý : O(cf(n)) = O(F(n)) với c là hằng số

VD: O(n2/2) = O(n2)

Ví dụ 3.1 : Tìm độ phức tap của giải thuật tính giá trị ex theo công thức gần đúng sau:

ex =1 + x/1! + x2/2! + . . . + xn/n! với x và n cho trước.

Void EXP1

{

1. Scanf(x); S = 1;
2. For (int i=1, i <= n, i++ )

{

p = 1;

For (int j=1, j <= i, j++ )

p = p \* x/j;

S = S + p;

}

}

Ta có thể coi phép toán tích cực ở đây là phép : p = p \* x/j;

Và nó được thực hiện : 1 + 2 + . . . + n = n(n-1)/2 lần

⇒ Thời gian thực hiện giải thuật là : T(n) = O(n2).

Cũng trường hợp tính ex ta có thể biểu diễn giải thuật theo cách khác (dựa vào số hạng trước để tính số hạng sau):

x2/2! = x/1! \* x/2; . . .; xn/n! = xn - 1 /(n - 1)! \* x/n;

Giải thuật có thể được viết :

Void EXP2

{

1. Scanf(x); S = 1; p = 1;
2. For (int i=1, i <= n, i++ )

{

p = p \* x/i;

S = S + p;

}

}

Trường hợp này thì thời gian thực hiện giải thuật lại là : T(n) = O(n) vì phép p \* x/i chỉ được thực hiện n lần.

Chú ý: Trong thực tế có những trường hợp thời gian thực hiện giải thuật không chỉ phụ thuộc vào kích thước của dữ liệu, mà còn phụ thuộc vào chính tình trạng của dữ liệu đó nữa.

Ví dụ 3.2: Cho một vec tơ V có n phần tử, xác định thời gian thực hiện giải thuật tìm trong V một phần tử có giá trị bằng X cho trước.

Void Tim;

{

1. Found = false; //Biến logic báo hiệu ngừng khi tìm thấy

i = 1;

2. While (i <= n and not Found )

if (V[i] = = X )

{

Found = true; k = i;

Cout << k;

}

Else

i = i + 1;

}

Ta coi phép toán tích cực ở đây là phép so sánh V[i] với X. Có thể thấy số lần phép toán tích cực này thực hiện phụ thuộc vào chỉ số i mà V[i] = X. Trường hợp thuận lợi nhất xảy ra khi X bằng V[1] một lần thực hiện.

Trường hợp xấu nhất khi X bằng V[n] hoặc không tìm thấy: n lần thực hiện.

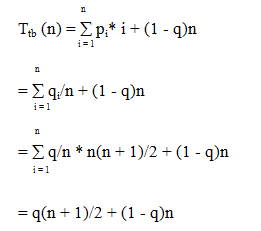
Vậy : Ttốt = O(1)

Txấu = O(n)

Lúc này ta phải xác định thời gian trung bình thực hiện giải thuật. Giả thiết khả năng xác suất X rơi đồng đều với mọi phần tử của V. Ta có thể xét như sau:

Gọi q là xác suất để X rơi vào một phần tử nào đó của V thì xác suất để X rơi vào phần tử V[i] là : pi\* = q/n

Còn xác suất để X không rơi vào phần tử nào sẽ là 1 - q. Khi đó ta sẽ xác định được thời gian thực hiện trung bình:



Nếu q = 1 ( nghĩa là luôn tìm thấy) thì Ttb (n) = (n + 1)/2

Nếu q = 1/2 (khả năng tìm thấy và không tìm thấy xác suất bằng nhau) thì Ttb = (n + 1)/4 + n/2 = (3n + 1)/4

Cả hai trường hợp đều dẫn đến cùng một kết quả là T(n) = O(n).