### 2.2. XÁC ĐỊNH ĐỘ PHỨC TẠP TÍNH TOÁN CỦA GIẢI THUẬT

Việc xác định độ phức tạp tính toán của một giải thuật bất kỳ có thể rất phức tạp. Tuy nhiên, trong thực tế, đối với một số giải thuật ta có thể phân tích bằng một số quy tắc đơn giản:

**2.2.1. Quy tắc cộng**

Nếu đoạn chương trình P1 có thời gian thực hiện T1(n) =O(f(n)) và đoạn chương trình P2 có thời gian thực hiện là T2(n) = O(g(n)) thì thời gian thực hiện P1 rồi đến P2 tiếp theo sẽ là

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

Chứng minh:

T1(n) = O(f(n)) nên  n1 và c1để T1(n)  c1.f(n) với  n  n1. T2(n) = O(g(n)) nên  n2 và c2 để T2(n)  c2.g(n) với  n  n2. Chọn n0 = max(n1, n2) và c = max(c1, c2) ta có:

Với  n  n0:

T1(n) + T2(n)  c1.f(n) + c2.g(n)  c.f(n) + c.g(n)  c.(f(n) + g(n))  2c.(max(f(n), g(n))). Vậy T1(n) + T2(n) = O(max(f(n), g(n))).

**2.2.2. Quy tắc nhân**

Nếu đoạn chương trình P có thời gian thực hiện là T(n) = O(f(n)). Khi đó, nếu thực hiện k(n) lần đoạn chương trình P với k(n) = O(g(n)) thì độ phức tạp tính toán sẽ là O(g(n).f(n))

Chứng minh:

Thời gian thực hiện k(n) lần đoạn chương trình P sẽ là k(n)T(n). Theo định nghĩa:

 ck  0 và nk để k(n)  ck(g(n)) với  n  nk

 cT  0 và nT để T(n)  cT(f(n)) với  n  nT

Vậy với  n  max(nT, nk) ta có k(n).T(n)  cT.ck(g(n).f(n))

**2.2.3. Một số tính chất**

Theo định nghĩa về độ phức tạp tính toán ta có một số tính chất:

a) Với P(n) là một đa thức bậc k thì O(P(n)) = O(nk). Vì thế, một thuật toán có độ phức tạp cấp đa thức, người ta thường ký hiệu là O(nk)

b) Với a và b là hai cơ số tuỳ ý và f(n) là một hàm dương thì logaf(n) = logab.logbf(n). Tức là: O(logaf(n)) = O(logbf(n)). Vậy với một thuật toán có độ phức tạp cấp logarit của f(n), người ta ký hiệu là O(logf(n)) mà không cần ghi cơ số của logarit.

c) Nếu một thuật toán có độ phức tạp là hằng số, tức là thời gian thực hiện không phụ thuộc vào kích thước dữ liệu vào thì ta ký hiệu độ phức tạp tính toán của thuật toán đó là O(1).

d) Một giải thuật có cấp là các hàm như 2n, n!, nn được gọi là một giải thuật có độ phức tạp hàm mũ. Những giải thuật như vậy trên thực tế thường có tốc độ rất chậm. Các giải thuật có cấp là các hàm đa thức hoặc nhỏ hơn hàm đa thức thì thường chấp nhận được.

e) Không phải lúc nào một giải thuật cấp O(n2) cũng tốt hơn giải thuật cấp O(n3). Bởi nếu như giải thuật cấp O(n2) có thời gian thực hiện là 1000n2, còn giải thuật cấp O(n3) lại chỉ cần thời

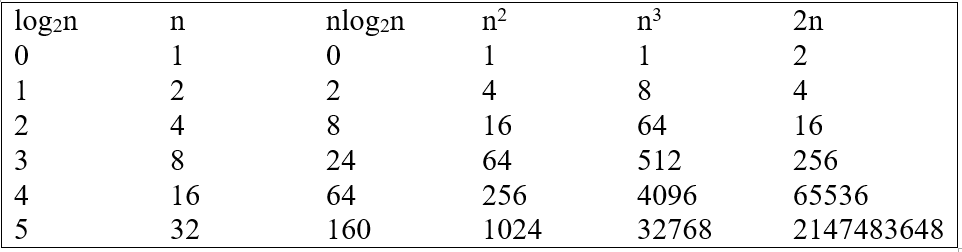
gian thực hiện là n3, thì với n < 1000, rõ ràng giải thuật O(n3) tốt hơn giải thuật O(n2). Trên đây là xét trên phương diện tính toán lý thuyết để định nghĩa giải thuật này "tốt" hơn giải thuật kia, khi chọn một thuật toán để giải một bài toán thực tế phải có một sự mềm dẻo nhất định.

f) Cũng theo định nghĩa về độ phức tạp tính toán

Một thuật toán có cấp O(1) cũng có thể viết là O(logn) Một thuật toán có cấp O(logn) cũng có thể viết là O(n) Một thuật toán có cấp O(n) cũng có thể viết là O(n.logn) Một thuật toán có cấp O(n.logn) cũng có thể viết là O(n2) Một thuật toán có cấp O(n2) cũng có thể viết là O(n3) Một thuật toán có cấp O(n3) cũng có thể viết là O(2n)

Vậy độ phức tạp tính toán của một thuật toán có nhiều cách ký hiệu, thông thường người ta chọn cấp thấp nhất có thể, tức là chọn ký pháp O(f(n)) với f(n) là một hàm tăng chậm nhất theo n.

Dưới đây là một số hàm số hay dùng để ký hiệu độ phức tạp tính toán và bảng giá trị của chúng để tiện theo dõi sự tăng của hàm theo đối số n.



Ví dụ:

Thuật toán tính tổng các số từ 1 tới n:

Nếu viết theo sơ đồ như sau:

Input n;

S := 0;

for i := 1 to n do S := S + i;

Output S;

Các đoạn chương trình ở các dòng 1, 2 và 4 có độ phức tạp tính toán là O(1).

Vòng lặp ở dòng 3 lặp n lần phép gán S := S + i, nên thời gian tính toán tỉ lệ thuận với n. Tức là độ phức tạp tính toán là O(n).

Vậy độ phức tạp tính toán của thuật toán trên là O(n). Còn nếu viết theo sơ đồ như sau:

Input n;

S := n \* (n - 1) div 2; Output S;

Thì độ phức tạp tính toán của thuật toán trên là O(1), thời gian tính toán không phụ thuộc vào n.

**2.2.4. Phép toán tích cực**

Dựa vào những nhận xét đã nêu ở trên về các quy tắc khi đánh giá thời gian thực hiện giải thuật, ta chỉ cần chú ý đến một phép toán mà ta gọi là phép toán tích cực trong một đoạn chương trình. Đó là **một phép toán trong một đoạn chương trình mà số lần thực hiện không ít hơn các phép toán khác**.

Xét hai đoạn chương trình tính ex bằng công thức gần đúng:



Chương trình 1: Tính riêng từng hạng tử rồi cộng lại

program Exp1;

var

i, j, n: Integer; x, p, S: Real;

begin

Write('x, n = '); ReadLn(x, n); S := 0;

for i := 0 to n do

begin

p := 1;

for j := 1 to i do p := p \* x / j;

S := S + p;

end;

WriteLn('exp(', x:1:4, ') = ', S:1:4);

end.

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

Số lần thực hiện phép toán này là n.

Vậy độ phức tạp tính toán của thuật toán là O(n).

Chương trình 2: Tính hạng tử sau qua hạng tử trước

program Exp2;

var

i, n: Integer;

x, p, S: Real;

begin

Write('x, n = '); ReadLn(x, n);

S := 1;

p := 1;

for i := 1 to n do

begin

p := p \* x / i;

S := S + p;

end;

WriteLn('exp(', x:1:4, ') = ', S:1:4);

end.

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

Số lần thực hiện phép toán này là: 0 + 1 + 2 + … + n = n(n-1)/2 lần.

Vậy độ phức tạp tính toán của thuật toán là O(n2)