**BÀI 2: ĐÁNH GIÁ ĐỘ PHỨC TẠP THUẬT TOÁN**

Nhóm:

1. Trần Thị Thu Hằng(17/10).
2. Vũ Minh Hằng.
3. Lâm Thị Hằng.
4. Trần Mạnh Hùng.

**100**. g(n) = ; f(n) = 6n

lim n -> ∞ = lim n -> ∞ = lim n -> ∞ = 0

Vậy f(n) = O(g(n)).

**101**. f(n) = n +2; g(n) =

lim n -> ∞ = lim n -> ∞ = lim n -> ∞ + lim n -> ∞ = 0

Vậy f(n) = O(g(n)).

**102**. f(n) = n+logn; g(n) = n

lim n -> ∞ = lim n -> ∞ = lim n -> ∞

lim n -> ∞ = lim n -> ∞

= lim n -> ∞ = 0

Vậy f(n) = O(g(n)).

**103**. f (n) = n2 + 3n + 4; g(n) = n3

lim n -> ∞ = lim n -> ∞ = lim n -> ∞ = 0

Vậy f(n) = O(g(n)).

**104**.

===0.

Do đó: =O(

**105.**

=+) = ∞+0 = ∞.

Vậy n+=Ω(

**106.**

= ∞.

Vậy: =Ω(log n).

**107.**

= =.

Vậy ≠ O(Ω().

**127**. f(n) = ; g(n) = 6n + 7

lim n -> ∞ = lim n -> ∞ = lim n -> ∞ = ∞

Vậy f(n) = Ω(g(n)).

**128**. f(n) =

lim n -> ∞ = lim n -> ∞ = lim n -> ∞

= lim n -> ∞ = 0

Vậy f(n) = O(g(n)).

**129**. f(n) = n g(n) = n2 – n

lim n -> ∞ = lim n -> ∞ = lim n -> ∞ = 0

Vậy f(n) = O(g(n)).

**130**. f(n) = n+ n; g(n) = 4nlog(n2+1)

lim n -> ∞ lim n -> ∞ = lim n -> ∞

= lim n -> ∞ = lim n -> ∞ = 0

Vậy f(n) = O(g(n)).

**131.**

==0.

Vậy n+3=Ω(

**132.**

= = =0.

Vậy: =Ω(

**145.**

Rõ ràng ta có: =) = 0.

Vậy, ta kết luận:=O(

**146.**

Ta cần chứng minh c (1)

Chứng minh bằng qui nạp:

* k=1: (1) đúng.
* Giả sử (1) đúng với k=t. Ta cầng chứng minh (1) đúng với k=t+1. Thay k=t+1, ta xét:

= = ) = 0.

Vậy (1) đúng.

**147.**

Ta có: .

Do đó: .

**148.**

Ta có: =∞.

Do đó: n! = Ω().

**149.**

Ta có: với mọi số n≥2.

Do đó: 0 → khi n→∞.

Vậy

Hay :

**150.**

Theo chứng minh ở bài 149, thì là sai.

**151.**

Đặt , với mọi n≥2,

Ta có: 0< =< <→0, khi a→∞.

Do đó: =0.

Vậy

**152.**

Theo chứng minh ở bài 151, ta cũng thu được là sai.

**153.**

Đặt : (n-1)!=a. Do đó: n!= a.n.

Ta xét: =

Mà 0 <→0 khi n→∞, hay = 0.

Vậy (n!)!=O ( là đúng.

**154.**

Theo những gì đã chứng minh ở bài 154 thì ta có:

(n!)!=Ω( là sai.

**155.**

Ta có : < =,

[] <, với mọi n nguyên dương.

Kết hợp với định ngĩa của O-lớn, thì

=O ().

=O ([]).

Vậy: O () = O ([]).

**Bài 1**: Xây dựng thuật toán sắp xếp chọn theo ý tưởng sau: sắp xếp n số lưu trong mảng A bằng cách tìm phần tử nhỏ nhất của A và đổi nó với phần tử A[1]. Sau đó tìm phần tử nhỏ thứ hai của A, và đổi nó với A[2]. Tiếp tục với n-1 phần tử của A.

1. Viết mã giả cho thuật toán này.
2. Bất biến của vòng lặp là gì?
3. Vì sao chỉ cần thực hiện với n-1 phần tử đầu tiên, thay cho cả n phần tử.
4. Đưa ra đánh giá thời gian thực hiện thuật toán trong trường hợp tốt nhất và xấu nhất theo kí hiệu O.

BL :

1. Mã giả:

Function selection sort A[1…n]

Comment sort A[1], …, A[n] into nondecreasing order

1. for i=1 to n-1 do
2. k = i;
3. min = A[i];
4. for j=i+1 to n do
5. if A[j] < min then (min = A[j]) ^ ( k = j );
6. A[k] = A[i];
7. A[i] = min;
8. return(A)
9. Bất biến vòng lặp : các phần tử A[1], …, A[i] được sắp xếp theo thứ tự không giảm ở bước thứ i.
10. Chỉ cần thực hiện với n-1 phần tử đầu tiên thay cho cả n phần tử vì:

Dãy n phần tử có n-1 phần tử đã được sắp xếp theo thứ tự không giảm thì đương nhiên phần tử còn lại cũng có thứ tự trong dãy mà không làm thay đổi tính chất không giảm.

1. Đánh giá thời gian thực hiện thuật toán:

* Trường hợp tốt nhất: dãy có thứ tự không giảm

Ta thực hiện n lần lệnh gán ở các bước 2, 3, 6, 7.

→ T(n) = O(4n) = O(n).

* Trường hợp xấu nhất: Dãy có thứ tự không tăng

Ở mỗi bước thứ i, ta tìm min của dãy A[i+1],…, A[n]. Thực hiện từ i=1 tới i= n-1 cần thời gian:

T(n) = (n-1) + (n-2) +…+1 = n(n-1)/2 = O(n2).

**Bài 2**: Xem xét bài toán tìm kiếm:

Vào: Một dãy n số A = <a1, a2, …, an> và một giá trị v.

Ra: Một chỉ số i sao cho v = A[i] hoặc bằng 0 nếu v không xuất hiện trong A.

1. Viết mã giả cho thuật toán này.
2. Sử dụng bất biến vòng lặp để chứng minh thuật toán trên là đúng đắn.
3. Đưa ra đánh giá thời gian thực hiện thuật toán trong trường hợp tốt nhất và xấu nhất theo kí hiệu O.

BL :

1. Mã giả:

Function Find(A[1…n], v)

Comment return i if v = A[i] else return 0

1. k = 0;// giá trị kiểm tra xem v có trong dãy không
2. for i=1 to n do
3. if A[i] = v then return(k=i);
4. return(k)

1. Chứng minh thuật toán trên là đúng đắn bằng bất biến vòng lặp:

* Bất biến vòng lặp: Bước thứ i, k=0 nếu v không có trong A[1],…, A[i]. Và, k=i nếu v có trong A[1],…, A[i].
* Khởi tạo: k=0 vì chưa duyệt phần tử nào của mảng. Đúng!
* Duy trì:

+ Lần duyệt thứ i: trả lại k=i và kết thúc vòng lặp nếu A[i] = v. Ngược lại, k=0 thì thực hiện vòng lặp tiếp theo.

+ Lần thứ i+1, trả lại k=i+1 nếu A[i+1] = v. Ngược lại, k=0.

* Kết thúc:

+ i=j, ta có k=j vì A[j] = v. Đã tìm được v trong mảng và trả lại vị trí của nó.

+ Hoặc i = n+1, k=0. Lúc này ta đã duyệt n phần tử của mảng, và đó cũng là toàn bộ phần tử của mảng. Kết luận, v không có trong mảng.

1. Đánh giá thời gian thực hiện thuật toán:

* Trường hợp tốt nhất: v = A[1]

Vòng lặp dừng lại ngay lần duyệt đầu tiên. Thời gian thực hiện: T(n) = O(1).

* Trường hợp xấu nhất: v = A[n] hoặc v không có trong mảng

Vòng lặp thực hiện n lần. Thời gian thực hiện: T(n) = O(n).

**Bài 259** (trang 50 – Ian Parberry – Problems on Algorithms)

Chứng minh thuật toán tính tổng các phần tử trong mảng A[1…n] sau đây là đúng.

Function sum(A)

Comment Return ∑A[i], i=1,…,n

1. s = 0;
2. for i=1 to n do
3. s = s + A[i];
4. return(s)

BL:

* Bất biến vòng lặp : Si = sum (0,A[1], …, A[i])
* Khởi tạo: S0 = 0 – chưa duyệt phần tử nào cảu mảng. → Đúng.
* Duy trì: Si  = sum(0,A[1], …, A[i])

Si+1 = sum(0,A[1], …, A[i], A[i+1])

* Kết thúc: i = n+1, ta nhận được tổng của n phần tử trong mảng. Và đó cũng là toàn bộ các phần tử trong mảng. Do đó, Sn = sum(0, A[1],…,A[n]).

**Bài 260** (trang 50 – Ian Parberry – Problems on Algorithms )

Chứng minh thuật toán tìm giá trị lớn nhất trong mảng A[1…n] sau đây là đúng.

Function max(A)

Comment Return max A[1],…, A[n]

1. m = A[1];
2. for i=2 to n do
3. if A[i] > m then m = A[i];
4. return(m)

BL :

* Bất biến vòng lặp: Mi = max(A[1],…, A[i])
* Khởi tạo: M1 = max(A[1]) \_ luôn đúng.
* Duy trì: Mi = max(A[1],…, A[i])

Mi+1 = max(A[1],…, A[i+1])

* Kết thúc: i = n+1, ta nhận được max của n phần tử. Và, đó cũng là tất cả phần tử của mảng. Do đó, Mn = max(A[1],…, A[n]).

**Bài 261** (trang 50 – Ian Parberry\_Problems on Algorithms )

Chứng minh tính đúng đắn của thuật toán sắp xếp nổi bọt sau. Các giá trị được sắp xếp cho trong mảng A[1…n]

Procedure bubblesort(A[1…n])

Comment Sort A[1], …, A[n] thành dãy không giảm

1. for i=1 to n-1 do
2. for j=1 to n-i do
3. if A[j] > A[j+1] then
4. swap A[j] with A[j+1];

BL:

* Bất biến vòng lặp: Các phần tử A[n-i],…, A[n] được sắp xếp theo thứ tự không giảm.
* Khởi tạo: i=1, thuật toán đẩy phần tử lớn nhất của mảng A[1],…, A[n] vào vị trí A[n]. Đúng vì mảng luôn tồn tại giá trị lớn nhất.
* Duy trì:

+ Lần thứ i, tìm giá trị lớn nhất trong dãy A[1]…A[n-i] xếp vào A[n-i] ta được A[n-i],…,A[n] sắp xếp theo thứ tự không giảm.

+ Lần thứ i+1, tìm giá trị lớn nhất trong dãy A[1]…A[n-i-1] xếp vào vị trí A[n-i-1] ta được A[n-i-1], A[n-i], …, A[n] sắp xếp theo thứ tự không giảm.

* Kết thúc: i=n, dãy có các phần tử A[2],…, A[n] sắp xếp theo thứ tự không giảm. Do đó, phần tử A[1] đương nhiên cũng ở đúng vị trí của nó. → Dãy có n phần tử được sắp xếp theo thứ tự không giảm và đó cũng là tất cả các phần tử của mảng.

**Bài 262** (trang 51 - Ian Parberry\_Problems on Algorithms )

Chứng minh tính đúng đắn của thuật toán sau: Đầu vào là một xâu S[1…n] và một xâu P[1…m] với 1 <= m <= n. Thuật toán đánh dấu vị trí đầu tiên mà P xuất hiện trong S, tức là l = p nếu S[p…p+m-1] = P, và l = n-m+1 nếu xâu P không có trong xâu S.

Function match(P, S, n, m)

Comment Find the pattern P[0…m-1] in string S[1…n]

1. l = 0; match = false;
2. while(l <= n-m) ^ ┐matched do
3. l = l+1;
4. r = 0; match = true;
5. while ( r < m) ^ matched do
6. matched = matched ^ ( P[r] = S[l+r] )
7. r = r+1;
8. return(l)

BL:

* Bất biến vòng lặp: 1 <= l <= n-m+1 và matched

+ l = i và matched = false : chưa tìm thấy xâu P xuất hiện trong xâu S.

+ l = i và matched = true : tìm thấy xâu P xuất hiện trong xâu S kể từ vị trí thứ i.

+ l = n-m+1 : xâu P không có trong xâu S.

* Khởi tạo: l = 0, matched = false : chưa duyệt phần tử nào của xâu P và S. Đương nhiên kết luận P không có trong S.
* Duy trì :

+ Lần thứ i ( 0 < i < n-m) : Nếu l=i và matched = true thì trả lại giá trị l=i\_là vị trí mà xâu P bắt đầu xuất hiện trong xâu S. Nếu l=i và matched = false thì thực hiện lần lặp tiếp theo.

+ Lần thứ i+1: Nếu l=i+1 và matched = true thì trả lại giá trị l=i+1 là vị trí mà P bắt đầu xuất hiện trong S. Nếu l=i+1 và matched=false thì thực hiện lần lặp tiếp theo.

* Kết thúc:

+ l = i và matched = true : nếu P có trong S kể từ vị trí thứ i.

+ l = n-m+1, kết luận xâu P không có trong xâu S.

**Bài 263** ( trang 51 – Ian Parberry\_Problems on Algorithms )

Chứng minh rằng thuật toán cộng ma trận sau là đúng.

Procedure matmultiply(Y, Z, n)

Comment multiplies nxn matrices YZ

1. for i=1 to n do
2. for j=1 to n do
3. X[i,j] = 0;
4. for k=1 to n do
5. X[i,j] = X[i,j] + Y[i,k]\*Z[k,j];
6. return(X)

BL:

* Bất biến vòng lặp: X[i,j] = sum( Y[i,k] \* Z[k,j] ) , với k = 1,…,n. Và thuật toán sum là đúng đắn theo chứng minh bài 259.
* Khởi tạo: X[i,j] = 0 vì chưa duyệt phần tử nào. Đúng.
* Duy trì:

+ X[i,j] = sum ( Y[i,k] \* Z[k,j] ) \_ Tính tổng của từng phần tử hàng thứ i của Y nhân với cột thứ j của Z.

+ X[i,j+1] = sum ( Y[i,k] \* Z[k,j+1] ) \_ Tính tổng của từng phần tử hàng thứ i của Y nhân với từng phần tử cột thứ j+1 của Z.

* Kết thúc: i=n+1, j=n+1 : đã duyệt hết n hàng của ma trận Y, n cột của ma trận Z và tìm được nxn phần tử của ma trận tích X. Đó chính là ma trận cần tính.