**BÀI TẬP SỐ 2**

**ĐỘ PHỨC TẠP CỦA THUẬT TOÁN**

**🙢🕮🙠**

*Thành viên nhóm:*

Hoàng Thị Quỳnh Hoa

Dư Thị Lan Hương

Nguyễn Thị Huyên

Đỗ Thị Duyên

***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 *A*[2]. Tiếp tục với  phần tử của *A*.

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

***Giải***

***Mã giả:***

**Selectsort(A,n)**{

for(i=1 to n-1){

k=i;

x=A[i];

for(j=i+1 to n){

if(A[j]<x){

x=A[j]; k=j;

}

}

A[k]=A[i];

A [i]=x;

}

}

**Bất biến vòng lặp:**

* Vòng lặp bên trong : lần lặp thứ k : =min(A[i+1],….,A[i+k])
* Vòng lặp bên ngoài : lần lặp thứ j dãy các phần tử A[1],…A[j] đã được sắp

**Chỉ cần thực hiện với n-1 phần tử đầu tiên vì:** Sau n - 1 bước thì dãy

A[1],…A[n-1] phần tử đã được sắp đúng thứ tự, trong cả mảng A[1],..,A[n] phần tử nhỏ thứ i đã đứng đúng vị trí A[i] như vậy A[n] sẽ lớn hơn các phần tử đằng trước và ta không cần sắp nữa.

**Đánh giá thời gian thực hiện:**

* Số lần so sánh: Độc lập với trạng thái ban đầu của dãy

 Số lần so sánh trong trường hợp xấu nhất và số lần so sánh trong trường hợp tốt nhất là như nhau :

Tại bước thứ i của vòng lặp ngoài cần n-i lần so sánh.Ta có:

= (n - 1) + (n - 2) + (n - 3) +…+ (n - (n - 1))

== O()

* Số phép gán:
* *Trường hợp tốt nhất* : Dãy ban đầu đã sắp đúng thứ tự trong mỗi bước so sánh không cần gán lại x = a[j]  tại mỗi bước của vòng lặp for bên ngoài chỉ cần 4 phép gán là k = i; x = a[i]; a[k] = a[i]; a[i] = x.

Số phép gán trong trường hợp tốt nhất = O(n)

* *Trường hợp xấu nhất* :Dãy ban đầu có thứ tự ngược=> quá trình tìm min ở mỗi lần so sánh phải thực hiện 2 phép gán lại giá trị k = j; x = a[j];

Bước 1: cần n - 1 lần so sánh và n-1 lần gán lại

Bước 2 : Do đã chuyển chỗ A[1] và A[n] nên chỉ còn (n - 2) – 1= n-3 lần gán lại

 Tóm lại: Số phép gán trong trường hợp xấu nhất là:

= O(

Vậy thời gian thực hiện thuật toán trong trường hợp tốt nhất và xấu nhất là O(

***Bài 2:***Xem xét ***bài toán tìm kiếm*:**

## Vào: Một dãy *n* số *A* = <*a*1, *a*2, ...,*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*.

* Viết mã giả cho thuật toán này.
* Sử dụng bất biến của vòng lặp để chứng minh thuật toán là đúng đắn (Hãy đảm bảo rằng bất biến của vòng lặp thoả mãn đủ ba tính chất cần thiết).
* Đưa ra đánh giá thời gian thực hiện thuật toán trong trường hợp tốt nhất và xấu nhất theo kí hiệu O.

**Giải**

1. Viết mã giả cho thuật toán

**find**(A,n, v)

{

k = 0;

**for**(i = 1 **to** n)

{

**If** (a[i] == v)

{

k = i;

}

}

**return** (k);

}

Bất biến vòng lặp: Giá trị của ki :

***Khởi tạo***: k1 = 1 nếu v = A1 trong dãy (A1)

k1 = 0 nếu v khác A1 trong dãy (A1)



***Duy trì:*** Bất biến vòng lặp đúng trước 1 vòng lặp thì nó cũng đúng với vòng lặp tiếp theo

Tại vòng lặp thứ j : thì

Tại vòng lặp thứ j + 1:

+) Nếu thì

+) Nếu thì



***Kết thúc***: i = n+1 sau t lần lặp (t = n+1 – 1 = n)

Chứng minh thuật toán là đúng đắn:

Vòng lặp kết thúc khi

+) i vượt quá n, ví dụ: i = n+1. Thay n+1 cho i trong bất biến vòng lặp ta nhận được mảng con A[1...n] với . Trả về kn = 0 nếu v không xuất hiện trong mảng A hoặc trả về chỉ số i ( ) với v = A[i] thuộc dãy A

Nên thuật toán(Trả về 1 chỉ số i sao cho v = Ai hoặc bằng 0 nếu v không xuất hiện trong A) là đúng

* Đưa ra đánh giá thời gian thực hiện:

Thời gian thực hiện = Thời gian so sánh (v so sánh với A[i]) + thời gian thực hiện phép gán (k = i) nếu v = A[i]

1. Trường hợp tốt nhất:

Số có giá trị bằng v ở ngay vị trí thứ nhất trong dãy

f(n) = O(1)

1. Trường hợp xấu nhất

Số có giá trị bằng v ở vị trí của phần tử cuối cùng trong dãy hoặc trường hợp không có giá trị v trong dãy

f(n) = O(n)

***Bài 3:* Các bài 100 – 107 (trang 29); 127 – 132 (trang 30); 145 – 155(trang 31), giáo trình [2] (Ian Parberry – *Problems on Algorithms.pdf*)**

1. Bài 100 – 107: Xác định f(n) = O(g(n)) hoặc g(n) = O(f(n))

***Bài 100:***

Xét f (n) = , g(n) = 6n

Ta có

Vậy g(n) = O(f(n))

***Bài 101****:*

f (n) = n + , g(n) =

Ta có:

Vậy f(n) = O(g(n))

***Bài 102:***

f(n) = n + logn, g(n) =

Ta có:

= 0

Vậy f(n) = O(g(n))

***Bài 103:***

f(n) = , g(n) =

Ta có:

Vậy f(n) = O(g(n))

***Bài 104:***

f(n) = nlogn, g(n) =

Ta có

Vậy f(n) = O(g(n))

***Bài 105****:*

f(n) = n + logn, g(n) =

Ta có

Vậy g(n) = O(f(n))

***Bài 106:***

f(n) = , g(n) = logn + 1

Ta có:

Vậy g(n) = O(f(n))

***Bài 107:***

f(n) = 4nlogn + n, g(n) = ()/2

Ta có:

Vậy f(n) = O(g(n))

1. Bài 127 – 132: Xác định f(n) = O(g(n)), hoặc f(n) = Ω(g(n)), f(n) =

***Bài 127:*** f(n) = , g(n) = 6n + 7

Ta có:

Vậy g(n) = O(f(n)) => f(n) = Ω(g(n))

***Bài 128*:** f(n) = , g(n) = log(n+3)

Ta có:

Vậy g(n) = O(f(n)) => f(n) = Ω(g(n))

***Bài 129:*** f(n) = , g(n) =

Ta có:

Vậy f(n) = O(g(n))

***Bài 130:*** f(n) = n + , g(n) =

Ta có: = ∞

Vậy f(n) = Ω (g(n))

***Bài 131:*** f(n) = ()/(), g(n) = n + 3

Ta có:

=



Vậy f(n) = Ω (g(n))

***Bài 132:***  f(n) = , g(n) =

Ta có:

Vậy f(n) = Ω (g(n))

1. Bài 145 – 155:

***Bài 145:* Chứng minh rằng:** cn2 +d = O(2n), với mọi c, d thuộc R+.

*Bài làm:* Xét giới hạn: =0.

→ cn2  + d = O(2n).

***Bài 146:* Chứng minh rằng:** cnk + d = O(2n), với mọi c, d, k thuộc R+.

*Bài làm:* Xét giới hạn: =0.

→ cnk  + d = O(2n).

***Bài 147:* Chứng minh rằng:** 2n = O(n!).

*Bài làm:* Xét giới hạn: =0.

→2n  = O(n!).

***Bài 148:* Chứng minh rằng:** n! = Ω(2n).

*Bài làm:* Xét giới hạn: = ∞.

→n! = Ω(2n).

***Bài 149:* Có phải nlog n = O((log n)n)? Chứng minh cho câu trả lời của bạn.**

*Bài làm:*

f(n) = nlog n ; g(n) = (log n)n

Ta có: 0 <=<= =

Mà khi n→∞(ln 2<1), do đó →0 khi n→∞

Vậy

→ nlog n = O()

***Bài 150*: Có phải nlog n = Ω((log n)n)? Chứng minh cho câu trả lời của bạn.**

*Bài làm:*

Theo bài 149 → nlog n = Ω() là sai.

***Bài 151*: Có phải nlog log log n = O((log n)!)? Chứng minh cho câu trả lời của bạn.**

*Bài làm:*

f(n) = nlog log log n ;g(n) = (log n)!

Đặt: log n = a,

f(n) =

g(n) = a!

Ta có: <= <= = → ∞(a→∞)

= ∞.

→ nlog log log n = O((log n)!) là sai

***Bài 152:* Có phải nlog log log n = Ω((log n)!)? Chứng minh cho câu trả lời của bạn.**

*Bài làm:*

Theo bài 151 → nlog log log n = Ω((log n)!) là đúng.

***Bài 154:* Có phải (n!)! = Ω(((n-1)!)!)? Chứng minh cho câu trả lời của bạn.**

*Bài làm:*

f(n) = (n!)!

g(n) = ((n-1)!)

Xét giới hạn:

Đặt: (n-1)! = a, n! = na

Ta có: (a+1)(a+2)…(a+a-1) =

2a(2a+1)…(2a+a-1) =

(n-1)a[(n-1)a + 1]…[(n-1)a+a-1] = (n-1)a.aa +

→(a+1)(a+2)…an = ((n-1)!)3.a(n-1)a.a.n. = aa.a(n-1)a.a.n. = aan.n

= = ∞

Vậy: (n!)! = Ω(((n-1)!)!.(n-1)!n!).

***Bài 153: Bài 154:* Có phải (n!)! = Ω(((n-1)!)!)? Chứng minh cho câu trả lời của bạn.**

*Bài làm:*

Theo bài 154 → (n!)! = O(((n-1)!)!.(n-1)!n!) là sai.

***Bài 155:* Chứng minh hay bác bỏ:**

**O() = O() .**

*Bài làm:*

Xét: = )

=

= = = ∞

→O() ≠ O() .

***Bài 4:* Các bài 259 – 263 (trang 50 - 51)**

***Bài 259:*** Chứng minh thuật toán tính tổng các số hạng trong mảng A[1..n] dưới đây là đúng:

**Function**sum(A)

**Comment** Return

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

*Giải:*

* *Sử dụng bất biến vòng lặp:* Tại bước thứ i của vòng lặp = A[1]+A[2]+…+A[i]=
* Khởi tạo:Trước vòng lặp đầu tiên = 0 đúng vì chưa xét phần tử nào trong mảng.
* Duy trì: Nếu thì + A[i+1]=
* Kết thúc: i=n+1 sau n lần lặp

=

***Bài 260***: Chứng minh thuật toán tìm giá trị lớn nhất của mảng A[1…n] bên dưới:

**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 the m:= A[i]
4. **return** (m)

*Giải:*

* Bất biến vòng lặp: = max(,…,)
* Khởi tạo: = = max() - đúng
* Duy trì: =max(,…) thì = max(,) = max(,…)
* Kết thúc: i=n+1 sau t lần lặp (t = n+1-2+1=n)

= max(,…) = max(,…)

***Bài 261*:** Chứng minh tính đúng của thuât toán sắp xếp nổi bọt bên dưới:

**produce** bubblesort(A[1…n])

**comment** Sort A[1], A[2], ….A[n] into nondecreasing order

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];

*Giải*:

* *Bất biến vòng lặp bên trong*: Trước lần lặp thứ j ,A[j] luôn là phần tử lớn nhất trong dãy A[1…j] hay A[j]= min ().
  + Khởi tạo: Trước lần lặp đầu tiên j=1 vì vậy A[1] là giá trị lớn nhất của dãy chỉ gồm A[1].
  + Duy trì: Trước lần lặp thứ j A[j] là giá trị lớn nhất trong dãy A[1..j] sau đó qua các dòng 3,4 thì A[j+1] là phần tử lớn nhất trong dãy A[1….j+1] .
  + Kết thúc: Vòng lặp kết thúc khi j=n-i+1 và A[j] là phần tử lớn nhất trong dãy A[1…n-i+1] hay A[1..j].
* *Bất biến vòng lặp bên ngoài*: Trước khi bắt đầu lần lặp thứ i dãy con A[n-i+1,….,n] đã được sắp.
* Khởi tạo: Trước lần lặp đầu tiên i = 1 dãy 1 pần tử A[n] được sắp.
* Duy trì: Trước lần lặp thứ i dãy A[n-i+1,….,n] đã được sắp sau đó qua các dòng 2-4 dãy con A[n-i,…n] được sắp đúng thứ tự.
* Kết thúc: i = n ta có dãy A[1….n] đã đúng thứ tự.

***Bài 262:***

Chứng minh tính đúng đắn của thuật toán đối sánh mẫu (pattern-matching). Đầu vào bao gồm chuỗi S[1... n] và 1 mẫu P[0 ... m - 1] với 1 m n. Thuật toán trả về vị trí của mẫu P trong chuỗi S, l = p nếu S[p ... p + m - 1] = P, và l = n – m +1 nếu mẫu P không xuất hiện trong chuỗi 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 ) **do**
3. l
4. r
5. **while** ( r ) **do**
6. matched
7. r
8. **return** (l)

Sử dụng bất biến vòng lặp để chứng minh thuật toán trên là đúng.

**Bất biến vòng lặp:**

Vòng lặp while ngoài: Giá trị của luôn 0 n – m + 1

Vòng lặp while trong: Giá trị của luôn 0

**Vòng lặp trong**

**Khởi tạo**: r = 0 nên 0 đúng

**Duy trì**: Do tại vòng lặp thứ i: , vòng lặp while chưa kết thúc < m

nên tại vòng lặp thứ i + 1:

**Kết thúc**: Vòng lặp while dừng khi hoặc

+) => đúng

+) => và

0 đúng

**Vòng lặp ngoài:**

Tương tự chứng minh bất biến vòng lặp như vòng lặp trong.

Vòng lặp while ngoài kết thúc khi:

1. và
2. và
3. và
4. => => không có mẫu P[m] trong xâu S[n]

Khi không xuất hiện mẫu P[m] trong xâu S[n] thì

1. => có mẫu P[m] trong xâu S[n]

Có mẫu P[m] trong xâu S[n] =>

1. => và

Nên => mẫu P[m] xuất hiện cuối xâu S[n]

Vậy thuật toán được chứng minh.

***Bài 263:***Chứng minh thuật toán nhân ma trân bến dưới là đúng:

**produce** matmutiply(Y,Z,n)

**comment** multiplies n x n matrics Y Z

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

*Giải:*

* *Bất biến vòng lặp tại dòng 2*: trước lần lặp thứ j phần tử cột thứ j-1 của ma trận tích được tính.
  + Khởi tạo: trước lần lặp đầu tiên j=1 chưa xét phần tử nào của ma trận tích.
  + Duy trì: Trước lần lặp thứ j phần tử ở cột j-1 được tính.

X[i][j-1]= sau đó qua các dòng 3-5 phần tử ở cột thứ j của ma trận tích được tính X[i][j]=

* + Kết thúc: j = n + 1 phân tử ở cột thứ n của ma trận tích đã được tính.
* *Bất biến vòng lặp tại dòng 1:* Trước lần lặp thứ h toàn bộ các phần tử ở hàng 1 tới h-1 của ma trận tích được tính.
  + Khởi tạo: trước lần lặp đầu tiên i = 1 chưa có hàng nào được tính vì chưa xét phần tử nào của ma trận tích.
  + Duy trì: trước lần lặp thứ h phần tử ở hàng thứ 1 tới h-1 của ma trận tích được tính, sau đó qua các dòng 2-5 phần tử ở hàng h của ma trận tích được tính.
  + Kết thúc: i = n + 1 toàn bộ các phần tử ở hàng từ 1 tới n của ma trận tích được tính. Như vậy thuật toán là đúng.