E – LEARNING 1

Mon Tue Wed Thu Fri Sat Sun DATE

Câu 1. Chứng minh tính đúng đắn của thuật toán sắp xếp nổi bọt.

Ta sẽ sử dụng phương pháp quy nạp:

* Giả sử ta có mảng A gồm n phần tử
* Dùng Bubble sort sắp xếp mảng tăng dần

- Bước 1: Cơ sở quy nạp (khi K = 1)
  - Khi vòng lặp ngoài chạy vòng đầu tiên (k = 1)
  - Vòng lặp trong sẽ duyệt qua tất cả các cặp phần tử từ đầu đến cuối mảng (từ A[j] đến A[n-2])
  - Mỗi khi gặp một cặp phần tử A[j] và A[j+1] mà A[j] > A[j+1], thuật toán sẽ hoán đổi vị trí chúng
  - Khi vòng lặp kết thúc, phần tử lớn nhất trong mảng sẽ được đẩy về vị trí cuối cùng.

→ Như vậy: mệnh đề đúng với trường hợp cơ sở (k = 1)

- Bước 2: Giả thiết quy nạp (khi K = m)
  - Giả sử sau khi vòng lặp chạy được m lần (k = m)
  - m phần tử lớn nhất trong mảng đã nằm đúng vị trí của chúng theo thứ tự tăng dần ở cuối mảng.

DATE Mon Tue Wed Thu Fri Sat Sun

• Các phần tử này bao gồm A[n-m],... A[n-1] và được sắp xếp theo thứ tự tăng dần.

-Bước 3: Bước quy nạp (khi k = m+1)

• Xét vòng lặp ngoài thứ m+1

• Vòng lặp chỉ cần xét các phần tử chưa được sắp xếp (từ A[0] đến A[n-m-2])

• Cơ chế nổi bọt sẽ đảm bảo rằng phần tử lớn nhất trong đoạn này được đẩy xuống cuối đoạn đang xét, tức tới vị trí A[n-m-1]

• Khi vòng lặp m+1 kết thúc, lại có thêm một phần tử đứng đúng vị trí.

• Như vậy, sau m+1 lần lặp, m+1 phần tử của mảng A sẽ được sắp xếp tăng dần ở cuối mảng (A[n-m-1] < A[n-m] < ... < A[n-1])

-Kết luận: Qua ba bước trên, ta kiểm chứng bằng phương pháp quy nạp và cho thấy Bubble sort là đúng đắn.

Mon Tue Wed Thu Fri Sat Sun DATE

Câu 2. Phân tích và độ phức tạp thời gian của Quick sort.

- Là thuật toán sắp xếp dùng nguyên lí chia để trị. Để phân tích độ phức tạp thời gian, ta sử dụng phương pháp đệ quy với ba trường hợp: tốt nhất, trung bình và xấu nhất.

* TH tốt nhất: thuật toán Quick sort chia mảng thành hai nửa gần bằng nhau. Giả sử việc phân hoạch (Partition) một mảng có n phần tử tốn thời gian tuyến tính O(n). Ta lập được phương trình sau:

$$T(n) = \begin{cases} 1 & , n = 1 \\ 2T\left(\frac{n}{2}\right) + n & , n > 1 \end{cases}$$

Sau đó, ta khai triển phương trình trên để tìm quy luật

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

$$\Leftrightarrow T(n) = \left[2T\left(\frac{n}{4}\right) + \frac{n}{2}\right] + n = 4T\left(\frac{n}{4}\right) + 2n$$

$$\Leftrightarrow T(n) = 4\left[2T\left(\frac{n}{8}\right) + \frac{n}{4}\right] + 2n = 8T\left(\frac{n}{8}\right) + 3n$$

Tổng quát sau lần khai triển:

$$T(n) = 2^i T\left(\frac{n}{2^i}\right) + i.n$$

Điều kiện dừng: Quá trình dừng lại khi mảng con còn một phần tử

$$\Leftrightarrow \frac{n}{2^i} = 1$$

$$\Leftrightarrow n = 2^i$$

$$\Rightarrow i = \log_2(n)$$

Kết hợp và xác định độ phức tạp

$$T(n) = 2^{\log_2(n)} . T(1) + \log_2 n . n = n \log n$$

Vì vậy độ phức tạp thời gian trong trường hợp tốt nhất là $O(n \log n)$

* TH trung bình: Xảy ra khi chọn pivot không gây ra sự mất cân bằng quá lớn trong quá trình phân hoạch. khi đó:

$$T(n) = T(\alpha.n) + T[(1-\alpha).n] + c.n \text{ với } 0 < \alpha < 1$$

Ta dùng định lí Master theorem mở rộng thì:

- Vì $\alpha.n$ và $(1-\alpha)n$ bé hơn n, và tổng chi phí phân hoạch mỗi lần là tuyến tính theo n, nên ta có:

$$T(n) = O(n \log n)$$

* TH xấu nhất: xảy ra khi chọn pivot ở đầu hoặc cuối mảng

$$T(n) = T(n-1) + T(0) + n$$

khai triển: $T(n) = T(n-1) + n$

$\Leftrightarrow T(n) = T(n-2) + (n-1) + n$

$\Leftrightarrow T(n) = T(n-3) + (n-2) + (n-1) + n$

Tổng này là một cấp số cộng:

$$n + (n-1) + (n-2) + \dots + 1 = \frac{n(n+1)}{2}$$

$\Rightarrow$ Độ phức tạp $T(n) = O(n^2)$

Câu 3. Cho hệ thức truy hồi

$$a_n = \begin{cases} 1 & , n = 0 \\ 6 & , n = 1 \\ 6a_{n-1} - 9a_{n-2} & , n > 1 \end{cases}$$

a) Cài đặt hàm tìm phần tử thứ n của hệ thức trên bằng 2 cách:

```python
#i, Dùng đệ quy
def fibo_recursion(n: int) -> int:  3 usages
    if n == 0:
        return 1
    elif n == 1:
        return 6
    return 6 * fibo_recursion(n-1) - 9 * fibo_recursion(n-2)

n = int(input("Nhập số thứ tự muốn tìm theo hệ thức: "))
print(f"Kết quả của hệ thức khi cài đặt bằng đệ quy khi n bằng {n}: {fibo_recursion(n)} ")
```

```
Nhập số thứ tự muốn tìm theo hệ thức: 5
Kết quả của hệ thức khi cài đặt bằng đệ quy khi n bằng 5: 1458

Process finished with exit code 0
```

```python
#ii, KHông dùng đệ quy
def fibo_loops(n: int) -> int:  1 usage
    if n == 1:
        return 6
    elif n == 0:
        return 1
    a,b = 1, 6
    for _ in range(2, n + 1):
        a, b = b, 6*b - 9*a
    return b

n = int(input("Nhập số thứ tự muốn tìm theo hệ thức: "))
print(f"Kết quả của hệ thức khi cài đặt bằng vòng lặp khi n bằng {n}: {fibo_loops(n)} ")
```

```
Nhập số thứ tự muốn tìm theo hệ thức: 5
Kết quả của hệ thức khi cài đặt bằng vòng lặp khi n bằng 5: 1458

Process finished with exit code 0
```

b) Phân tích và đánh giá độ phức tạp của 2 hàm trên, nếu lựa chọn thì chọn cách nào ? Vì sao ?

* Cách cài đặt bằng đệ quy

```
def fibo_recursion(n: int) -> int:
    if n == 0:
        return 1
    if n == 1:
        return 6
    return 6 * fibo_recursion(n-1) - 9 * fibo_recursion(n-2)
```

- Ta tiến hành xây dựng phương trình đệ quy và giải để tìm độ phức tạp về thời gian.

$$T(n) = \begin{cases} 1 & n=1, n=0 \\ T(n-1)+T(n-2)+O(1) & n>1 \end{cases}$$

Ta biết : $T(n) \leq c.g(n)$ thì $T(n) = O(g(n))$

$$T(n) = T(n-1) + T(n-2) + O(1)$$

$\Leftrightarrow T(n) = T(n-1)+T(n-2)+O(1) \leq T(n-1)+T(n-1)+O(1)$

$\Leftrightarrow T(n) = T(n-1)+T(n-2) \leq 2T(n-1)$

$\Leftrightarrow T(n) = T(n-1)+T(n-2) \leq 2[2T(n-2)]$

$\Leftrightarrow T(n) = T(n-1)+T(n-2) \leq 4[2T(n-3)]$

$\Leftrightarrow \ldots \leq \ldots$

$\Leftrightarrow T(n) = T(n-1)+T(n-2) \leq 2^n . T(0)$

$\Rightarrow T(n) = O(2^n)$

- Về độ phức tạp không gian : sử dụng đệ quy, độ phức tạp không gian là $O(n)$

DATE Mon Tue Wed Thu Fri Sat Sun

* Cài đặt bằng vòng lặp

- Phân tích từng bước :

```
def fibo_loops(n: int) -> int:
    if n == 1:          ← O(1)
        return 6        ← O(1)
    if n == 0:          ← O(1)
        return 1        ← O(1)
    a,b = 1, 6          ← O(1)
    for _ in range(2, n + 1):   ← O( n+1-1-2+1)
        a,b = b, 6*b - 9*a
    return b            ← O( 1)
```

- Dùng Quy tắc nhân với vòng lặp và quy tắc cộng tìm nhanh phức tạp nhất, ta có độ phức tạp thời gian như sau :

$$T(n) = O(1) + O(1) + O(1) + O(1) \cdot O(n-1)$$
$$= O(n)$$

- Về độ phức tạp về không gian, chỉ dùng biến, không mảng, không đệ quy nên độ phức tạp là O(1)

Từ đó ta dễ dàng nhận thấy, nên chọn cách cài đặt bằng vòng lặp vì :

+ Độ phức tạp thời gian chỉ là $O(n)$ nhỏ hơn $O(2^n)$ của đệ quy

+ Độ phức tạp không gian là $O(1)$ tốt nhất

- Cách dùng đệ quy dễ hiểu, mã ngắn gọn bám sát toán học nhưng không phù hợp chạy thực tế!

c) Theo bạn còn cách nào tối ưu hơn? Vì sao?

+ Dùng công thức nghiệm tổng quát:

$$a_n = A \cdot r_1^n + B \cdot r_2^n$$

Với r_1 và r_2 là nghiệm của phương trình đặc trưng

$$r^2 - 6r + 9 = 0 \Leftrightarrow (r-3)^2 = 0 \Leftrightarrow r_1 = r_2 = 3$$

Nghiệm a_n có dạng: $a_n = (A + Bn) \cdot 3^n$

Dựa vào điều kiện ban đầu:

$a_0 = 1 = A$

$a_1 = 6 = (A+B) \cdot 3 \Rightarrow A = 1, B = 1 \rightarrow$ $a_n = (1+n) \cdot 3^n$

```python
def a_formula(n):  1 usage
    return (1 + n) * (3**n)
print(f"Giá trị tại vị trí thứ 5 là: {a_formula(5)}")
```

```
Giá trị tại vị trí thứ 5 là: 1458

Process finished with exit code 0
```

Rõ ràng chi phí chỉ là $O(1)$ → Tối ưu cả độ phức tạp và cài đặt.