Đại học quốc gia TP.HCM
Trường đại học công nghệ thông tin

Môn học: Phân tích và thiết kế thuật toán
CS112.N21.KHTN

Bài tập Phân tích độ phức tạp của thuật toán không đệ quy

Sinh viên thực hiện:
Trương Thanh Minh - 21520064
Lê Châu Anh - 21521821

Mục lục

1 Bài 1 **2**
1.1 Đề bài 2
1.2 Giải 2

2 Bài 2 **3**
2.1 Đề bài 3
2.2 Giải 3

3 Bài 3 **4**
3.1 Đề bài 4
3.2 Giải 4

1 Bài 1

1.1 Đề bài

The **range** of a finite nonempty set of n real numbers S is defined as the difference between the largest and smallest elements of S. For each representation of S given below, describe in English an algorithm to compute the range. Indicate the time efficiency classes of these algorithms using the most appropriate notation (O, Θ, Ω)

a. An unsorted array

b. A sorted array

c. A sorted singly linked list

d. A binary search tree

1.2 Giải

Nhìn chung, bài toán ở trên sẽ quy về bài toán tìm số lớn nhất và nhỏ nhất trong một mảng. Ta có thể thấy rằng, để tìm được số lớn nhất và số nhỏ nhất ở trong mảng thì cái mảng đó phải được sắp xếp.

a. **An unsorted array**

Để giải quyết bài toán này, ta có thể sử dụng một vòng lặp để duyệt qua các phần tử trong mảng để tìm phần tử có giá trị lớn nhất và phần tử có giá trị nhỏ nhất. Sau đó, ta có thể giải quyết bài toán bằng cách lấy giá trị của phần tử lớn nhất trừ cho giá trị của phần tử nhỏ nhất.

Do đó, độ phức tạp ở đây trong cả trường hợp tốt nhất, xấu nhất và trung bình là như nhau: $\Omega(n), \Theta(n), O(n)$.

b. **A sorted array**

Vì trong trường hợp mảng đã sắp xếp thì ta chỉ việc lấy phần tử lớn nhất, nhỏ nhất trong $O(1)$. Cuối cùng, ta có thể giải quyết bài toán bằng cách lấy phần tử cuối cùng trừ cho phần tử đầu tiên (nếu mảng sắp xếp tăng dần, và ngược lại trong trường hợp mảng được sắp xếp giảm dần).

Do đó cả 3 trường hợp tốt nhất, xấu nhất và trung bình đều có độ phức tạp như nhau: $\Omega(1), \Theta(1), O(1)$.

c. **A sorted singly linked list**

Trong danh sách đã liên kết đã sắp xếp cũng tương tự như trong trường hợp với mảng đã sắp xếp, ta đều có thể lấy được giá trị lớn nhất và nhỏ nhất trong $O(1)$. Vì trong bài này ta giả sử cấu trúc danh sách liên kết của ta sẽ có thêm một node để lưu lại node cuối cùng trong list. Cuối cùng, ta có thể giải quyết bài toán bằng cách lấy phần tử cuối cùng (tail) của list trừ cho phần tử đầu tiên của list (head) (nếu mảng sắp xếp tăng dần, và ngược lại trong trường hợp mảng được sắp xếp giảm dần).

Do đó cả 3 trường hợp tốt nhất, xấu nhất và trung bình đều có độ phức tạp như nhau: $\Omega(1), \Theta(1), O(1)$.

d. **A binary search tree**

Cây nhị phân tìm kiếm là cây mà khi ta xét một nút bất kỳ, giá trị của mọi node trên cây con trái **luôn nhỏ hơn** node đang xét và giá trị của mọi node trên cây con phải **luôn lớn hơn** node đang xét.

Khi đó, phần tử nhỏ nhất trong cây sẽ là phần tử trái nhất, còn phần tử lớn nhất sẽ là phần tử bên phải nhất. Khi đó, độ phức tạp của bài toán này sẽ là chiều cao của cây.

- **Trong trường hợp xấu nhất (O)**
  Cây nhị phân tìm kiếm của ta nó chỉ có một nhánh, tức là cây hoàn toàn lệch sang trái hoặc hoàn toàn lệch sang phải. Khi đó, độ phức tạp của ta sẽ là $O(n)$.
- **Trong trường hợp trung bình (Θ)**
  Trung bình, chiều cao của cây nhị phân tìm kiếm sẽ là $log(N)$. Do đó, thời gian thực thi của nó sẽ là $\Theta(log(N))$
- **Trong trường hợp tốt nhất (Ω)**
  Trường hợp tốt nhất của một cây nhị phân tìm kiếm là một cây nhị phân cân bằng. Khi đó, độ phức tạp trong trường hợp này là $\Omega(log(N))$

2 Bài 2

2.1 Đề bài

Lighter or heavier? You have $n > 2$ identical-looking coins and a two-pan balance scale with no weights. One of the coins is a fake, but you do not know whether it is lighter or heavier than the genuine coins, which all weigh the same. Design a $\Theta(1)$ algorithm to determine whether the fake coin is lighter or heavier than the others.

2.2 Giải

Ta có thể giải quyết bài toán này như sau:

- **Bước 1:** Đầu tiên, chia n đồng xu ban đầu vào 3 nhóm. Mỗi nhóm có $\frac{n}{3}$ đồng xu. Ta gọi các nhóm đó lần lượt là nhóm A, B, C. Trong trường hợp nếu n không chia hết cho 3 thì ta có thể chia mỗi nhóm A, B là có số đồng xu là phần nguyên của $\frac{n}{3}$, số xu còn lại sẽ bỏ vào cho nhóm C.
- **Bước 2:** Tiếp theo, ta thực hiện cân nhóm A với nhóm B. Khi đó, sẽ có 3 khả năng xảy ra:
  - A và B cùng khối lượng. Điều này có nghĩa đồng xu giả nằm trong nhóm C.
  - A nặng hơn B. Điều này có nghĩa đồng xu giả nặng hơn và nằm trong nhóm A. Hoặc đồng xu giả nằm trong nhóm B và nhẹ hơn.
  - B nặng hơn A. Điều này có nghĩa đồng xu giả nằm trong nhóm B và nặng hơn. Hoặc đồng xu giả nằm trong nhóm A và nhẹ hơn.

- **Bước 3:** Bất kể nhóm nào nặng hơn trong bước 2, ta cân 2 đồng xu của nhóm đó với nhau. Có 2 kết quả có thể xảy ra:
  - 2 đồng xu có cùng khối lượng: Điều này có nghĩa là đồng xu giả là đồng xu còn lại trong nhóm đó, và nó nặng hơn hoặc nhẹ hơn tùy thuộc vào nhóm nào nặng hơn ở bước 2.
  - Một trong 2 đồng xu nặng hơn: Điều này có nghĩa là đồng xu giả là đồng xu đó và nó nặng hơn hoặc nhẹ hơn tùy thuộc vào nhóm nào nặng hơn ở bước 2.
- **Bước 4:** Nếu ở bước 2, ta thấy rằng nhóm A và nhóm B có cùng khối lượng, thì ta tiến hành lặp lại bước 3 với các nhóm A và C hoặc B và C.

3 Bài 3

3.1 Đề bài

ALGORITHM $GE(A[0..n-1, 0..n])$

```
//Input: An n × (n + 1) matrix A[0..n − 1, 0..n] of real numbers
for i ← 0 to n − 2 do
    for j ← i + 1 to n − 1 do
        for k ← i to n do
            A[j, k] ← A[j, k] − A[i, k] ∗ A[j, i] / A[i, i]
```

a. Find the time efficiency class of this algorithm.

b. What glaring inefficiency does this pseudocode contain and how can it be eliminated to speed the algorithm up?

3.2 Giải

a. **Find the time efficiency class of this algorithm**

Độ phức tạp của thuật toán này là $O(n^3)$. Vì:

- Ở vòng for cho i, độ phức tạp là $O(n)$.
- Ở vòng for cho j, độ phức tạp là $O(n)$.
- Ở vòng for cho k, độ phức tạp là $O(n)$.

Mà mà vòng for này lồng vào nhau. Do đó, độ phức tạp của thuật toán này hiện tại là $O(n^3)$

b. **What glaring inefficiency does this pseudocode contain and how can it be eliminated to speed the algorithm up?**

Ở vòng for cho k, khi ta xét $k = i$:

$A[j,k] \leftarrow A[j,k] - A[i,k] * A[j,i]/A[i][i]$

$\Leftrightarrow A[j,i] \leftarrow A[j,i] - A[i,i] * A[j,i]/A[i,i]$

$\Leftrightarrow A[j,i] \leftarrow A[j,i] - A[j,i]$

$\Leftrightarrow A[j,i] \leftarrow 0$

Các trường hợp còn lại, tức là $k : i+1 \rightarrow n$

$A[j,k] \leftarrow A[j,k] - A[i,k] * A[j,i]/A[i][i]$

$\Leftrightarrow A[j,k] \leftarrow A[j,k] - A[i,k] * 0/A[i,i]$ (Vì ta thấy rằng, khi $k = i$, thì $A[j,i] = 0$).

$\Leftrightarrow A[j,k] \leftarrow A[j,k]$.

Tóm lại, ta thấy rằng, $\forall k \in [i+1, n]$, việc gán là vô ích. Do đó, ta có thể bỏ vòng for này. Ta có thể viết lại mã giả như sau:

for $i \leftarrow 0$ **to** $n-2$ **do**
  for $j \leftarrow i+1$ **to** $n-1$ **do**
    $A[j,i] = 0$
  end for
end for

Cuối cùng, sau khi loại bỏ những phần không cần thiết, độ phức tạp của ta còn $O(n^2)$.