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

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 Solution

The problem above can be rewritten as the problem of finding the largest and smallest elements in an array.

a. **An unsorted array**

- Algorithm: We can iterate over the array using for loop to find the largest and smallest elements. Finally, subtract the smallest element from the largest element.
- Time complexity: The same in both the best, worst and average cases: $\Omega(n)$, $\Theta(n)$, $O(n)$.

b. **A sorted array**

- Algorithm: Subtract the first element from the last element for ascending array and do the opposite for descending array.
- Time complexity: With a sorted array, take the largest/smallest element will take $O(1)$ time. Therefore, both the best, worst and average cases have the same time complexity: $\Omega(1)$, $\Theta(1)$, $O(1)$.

c. **A sorted singly linked list**

- Algorithm: We assume there is an extra node to store the last node in the single linked list. Then, we find the values of first element (head) and last element (tail). Finally, subtract the first value from the last value for ascending linked list and do the opposite for descending linked list.
- Time complexity: With a sorted singly linked list, take the largest/smallest element will take $O(1)$ time. Therefore, both the best, worst and average cases have the same time complexity: $\Omega(1)$, $\Theta(1)$, $O(1)$.

d. **A binary search tree**

Binary search tree is a node-based binary tree data structure which has the following properties:

- The left subtree of a node contains only nodes with keys lesser than the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
- The left and right subtree each must also be a binary search tree.

In this case, the smallest element is value of the leftmost node, the largest element is value of the rightmost node.

- Algorithm: Find values of the leftmost node and the rightmost node. Finally, subtract the first value from the last value.
- Time complexity:
  - **Worst case (O)**
    Binary search tree has only one branch, which means it completely skewed to the left/right. Then, time complexity = $O(n)$.
  - **Average case (Θ)**
    On average, the height of a binary search tree is $log(N + 1)$. Therefore, time complexity = $\Theta(log(N))$
  - **Best case (Ω)**
    Binary search tree is perfectly balanced. Height of the binary search tree becomes log(n). Therefore, time complexity = $\Omega(log(N))$.

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: Ta sẽ quy đổi số đồng xu (n) về một số chia hết cho 3 bằng cách sau:

- Nếu $n\%3 == 0$ thì không còn gì bàn cãi.
- Nếu $n\%3 == 1$ thì ta cần loại bỏ 1 đồng xu mà ta chắc chắn nó là đồng xu thật. Để làm được điều đó, ta có thể lấy ra 3 đồng xu bất kỳ(tạm gọi là x, y, z). Đầu tiên, ta cân x với y,
  - $x = y$ tức là cả hai đều là đồng xu thật, lúc này ta chỉ cần loại bỏ 1 trong 2 đồng xu bất kì.

– Trường hợp ngược lại, nếu x khác y, ta tiến hành cân x với z, nếu x bằng z thì ta chỉ cần loại bỏ một trong 2 đồng xu đó. Tuy nhiên nếu x khác z thì ta biết ngay đồng xu giả là đồng xu x (Giải quyết xong bài toán). Còn vấn đề đồng xu giả nặng hay nhẹ hơn thì tùy vào khi cân x với y.

- Nếu $n\%3 == 2$ thì ta cần loại bỏ 2 đồng xu thật. Lúc này để đơn giản, khỏi phải suy nghĩ nhiều. Ta có thể thực hiện giống trường hợp $n\%3 == 1$ nhưng ở đây ta sẽ loại 2 đồng xu thật chứ không phải như trường hợp đó.

Sau khi thực hiện xong bước trên, tức là đưa số đồng xu về một số chia hết cho 3. Ta có thể giải quyết bài toán còn lại một cách đơn giản 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.
- **Bước 2:** Ta chỉ cần cân tối đa 2 lần. Lần đầu tiên, ta cân A với B.
  - $A = B$, thì ta chắc chắn rằng đồng xu giả chỉ nằm trong nhóm C. Khi đó, ta chỉ cần cân A với C để biết đồng xu giả nặng hay nhẹ hơn.
  - A khác B thì lúc này ta cân A với C. Nếu $A = C$ thì đồng xu giả nằm trong B, ngược lại thì đồng xu giả nằm trong A. Còn về vấn đề đồng xu giả nặng hay nhẹ hơn thì ta có thể dễ dàng biết được trong 2 lần cân đó.

Tóm lại, sau bước giải trên, ta có thể dễ dàng thấy được rằng, thuật toán trên chỉ tốn tối đa **6 bước kiểm tra** (**4 bước** cho phần làm cho số đồng xu ban đầu về thành một số chia hết cho 3 + **2 bước** cho phần đồng xu giả nặng hay nhẹ hơn).

Do đó, độ phức tạp của thuật toán này chỉ nằm trong $\Theta(1)$.

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 ← 0 to n − 2 do
    for j ← 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 $O(n^2)$.