ĐẠI HỌC QUỐC GIA TP.HCM
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

CS112.N21.KHTN

BÀI TẬP TUẦN 1

PHÂN TÍCH ĐỘ PHỨC TẠP CỦA THUẬT TOÁN KHÔNG ĐỆ QUY

Nhóm : 14
GV hướng dẫn : Nguyễn Thanh Sơn

Tp.HCM ngày 19 tháng 3 năm 2023

1 Bài 1 (Bài 10 – trang 60)

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.

a. An unsorted array

b. A sorted array

c. A sorted singly linked list

d. A binary search tree

Trả lời

1.1 An unsorted array

Ta duyệt qua từng phần tử trong mảng và so sánh với giá trị hiện tại của biến lớn nhất. Nếu phần tử đó lớn hơn, ta cập nhật biến lớn nhất bằng giá trị của phần tử đó. Do đó, độ phức tạp thời gian của thuật toán là O(n).

```
range(arr):
    n = len(arr)
    min_val = float("inf")
    max_val = float("-inf")

    for i in range(n):
        if arr[i] < min_val:
            min_val = arr[i]
        if arr[i] > max_val:
            max_val = arr[i]

    return max_val - min_val
```

1.2 A sorted array

Ta có thể truy cập phần tử cuối cùng trong mảng để tìm giá trị lớn nhất. Do đó, độ phức tạp thời gian của thuật toán là $\Theta(1)$.

```
range_sorted(arr):
    return arr[-1] - arr[0]
in_val
```

1.3 A sorted singly linked list

Chúng ta cần duyệt qua toàn bộ danh sách liên kết để tìm phần tử nhỏ nhất và lớn nhất như đối với mảng chưa được sắp xếp. Vì vậy, độ phức tạp trung bình và độ phức tạp xấu nhất là $\Theta(n)$.

```
range_linked_list(head):
    if not head:
        return None

    min_val = head.val
    max_val = head.val

    curr = head.next
    while curr:
        if curr.val < min_val:
            min_val = curr.val
        if curr.val > max_val:
            max_val = curr.val
        curr = curr.next

    return max_val - min_val
```

1.4 A binary search tree

Độ phức tạp ở cây tìm kiếm nhị phân đã được sắp xếp là $\Theta(logn)$ vì mỗi lần ta duyệt xuống một tầng của cây thì số phần tử cần xét giảm đi một nửa. Vì vậy, để tìm phần tử lớn nhất hoặc nhỏ nhất, ta chỉ cần duyệt qua đường dẫn từ gốc đến lá thích hợp.

Tuy nhiên, độ phức tạp trong trường hợp xấu nhất sẽ là $\Theta(n)$, khi cây tìm kiếm nhị phân chỉ có một nhánh và các phần tử nằm trên cùng một nhánh đó. Trong trường hợp này, việc tìm phần tử lớn nhất hoặc nhỏ nhất sẽ tương đương với việc duyệt qua toàn bộ cây.

```
range_bst(root):
    if not root:
        return None

    min_val = root.val
    while root.left:
        min_val = root.left.val
        root = root.left
```

```
    max_val = root.val
    while root.right:
        max_val = root.right.val
        root = root.right

    return max_val - min_val
```

2 Bài 2 (Bài 11 - trang 61)

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.

Trả lời

Ta có thể giải bài toán như sau:

Bước 1: Chia n đồng xu thành 3 đống (n/3, n/3, phần còn lại), đặt 2 đống có số đồng xu bằng nhau lên 2 đĩa cân bên trái và phải, và để 1 đống bên ngoài.

Bước 2: So sánh 2 đống trên cân. Nếu bằng nhau, thì đồng xu giả ở đống bên ngoài. Cân đống bên ngoài với 1 trong 2 đống thật (có thể thêm 1 hoặc bớt 1 đồng xu cho số lượng bằng nhau). Nếu không bằng nhau, chuyển sang Bước 3.

Bước 3: Lấy đống nặng hơn từ Bước 2 và chia thành 2 đống bằng nhau. Đặt 1 đống lên mỗi đĩa cân bên trái và phải.

Bước 4: So sánh 2 đống trên cân. Nếu bằng nhau, đồng xu giả là đồng xu nằm trong đống nhẹ hơn của Bước 3. Nếu không bằng nhau, đồng xu giả là đồng xu nằm trên cân thuộc đống nặng hơn.

Với thuật toán này, ta chỉ cần cân đồng xu 2 lần và sử dụng đúng 3 đống đồng xu, nên độ phức tạp là $\Theta(1)$.

3 Bài 3 (Bài 11 - trang 69)

```
ALGORITHM GE(A[0..n - 1, 0..n])
   //Input: An n x (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 pseudo-code contain and how can it be eliminated to speed the algorithm up?

Trả lời

a. Vì có 3 vòng lặp chạy từ 0 đến n, với mỗi vòng lặp có độ dài n nên độ phức tạp thời gian của thuật toán GE là $O(n^3)$

b. Thuật toán GE trên vẫn có thể cải tiến được để tối ưu hơn. Cụ thể, ở trong vòng lặp thứ hai, ta có thể chỉ tính toán trên các phần tử từ vị trí i+1 trở đi, bởi vì các phần tử trước đó đã được biến đổi trong lần lặp trước đó.

Vì vậy, ta có thể thay vòng lặp thứ hai bằng:

```
for j <- i + 1 to n - 1 do
// chi tinh toan cac phan tu tu vi tri i+1 tro di
for k <- i + 1 to n do
A[j, k] <- A[j, k] - A[i, k] * A[j, i] / A[i, i]
```

Như vậy, trong vòng lặp này ta chỉ tính toán trên n - i - 1 phần tử thay vì n - i phần tử như ban đầu, giảm thiểu số lần tính toán và tối ưu thuật toán. Điều này giúp tránh phép chia và giảm độ phức tạp của thuật toán xuống còn $O(n^2)$.