ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA KHOA HỌC MÁY TÍNH

GIẢI BÀI TẬP

PHÂN TÍCH ĐỘ PHỨC TẠP THUẬT TOÁN ĐỆ QUY

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

Sinh viên thực hiện: Hà Văn Hoàng - 21520033

Võ Thị Phương Anh - 21522883

Thành phố Hồ Chí Minh, ngày 03 tháng 04 năm 2022

Mục lục

1. Bài tập 1: Tower of Hanoi (Tháp Hà Nội)

1. In the original version of the Tower of Hanoi puzzle, as it was published in the 1890s by Edouard Lucas, French mathematician, the world will end after 64 disks have been moved from a mystical Tower of Brahma. Estimate the number of years it will take if monks could move one disk per minute. (Assume that monks do not eat, sleep, or die.)

2. How many moves are made by the ith largest disk ($1 \leq$ i $\leq$ n) in this algorithm?

3. Find a non-recursive algorithm for the Tower of Hanoi puzzle and implement it in the language of your choice.

Dịch sang tiếng Việt:

1. Trong bản gốc của câu đố Tháp Hà Nội, được xuất bản vào những năm 1890 bởi nhà toán học người Pháp Edouard Lucas, thế giới sẽ kết thúc sau khi 64 chiếc đĩa được di chuyển khỏi Tháp Brahma thần bí. Ước tính số năm cần thiết nếu các nhà sư có thể di chuyển một đĩa mỗi phút. (Giả định rằng các nhà sư không ăn, không ngủ và không chết.)

2. Đĩa lớn thứ $i(1 \leq i \leq n)$ đã được di chuyển bao nhiêu lần trong thuật toán này?

3. Tìm một thuật toán không đệ quy cho câu đố Tháp Hà Nội và triển khai nó bằng ngôn ngữ lập trình bạn chọn.

Giải

1.1. Câu 1

Số bước cần thiết để các nhà sư di chuyển hết 64 đĩa là:

$$2^{64} - 1 = 18446744073709551615$$

Vậy số phút cần thiết để các nhà sư di chuyển hết 64 đĩa là:

18446744073709551615 phút

Suy ra số năm cần thiết để các nhà sư di chuyển hết 64 đĩa xấp xỉ là:

35072522765437 năm

1.2. Câu 2

Hãy nhận thấy rằng cho mỗi lần di chuyển của đĩa i, thuật toán đầu tiên sẽ di chuyển toàn bộ chồng đĩa nhỏ hơn nó sang một cọc khác (điều này yêu cầu một lần di chuyển của đĩa (i + 1)) và sau đó, sau khi di chuyển đĩa i, chồng đĩa nhỏ hơn này sẽ được di chuyển lên trên đĩa i (điều này lại yêu cầu một lần di chuyển của đĩa (i + 1)). Do đó, cho mỗi lần di chuyển của đĩa i, thuật toán sẽ di chuyển đĩa (i + 1) chính xác hai lần. Vì với i = 1, số lần di chuyển là $m(1) = 1$, ta có phương trình đệ quy sau đây cho số lần di chuyển của đĩa i (i $>$ 1) là: $m(i) = 2m(i-1)$.

Công thức giải phương trình đệ quy trên là: $m(i) = 2^{i-1}$ cho $i = 1, 2, ..., n$. (Cách đơn giản nhất để thu được công thức này là sử dụng công thức cho phần tử chung của một cấp số nhân.) Lưu ý rằng câu trả lời phù hợp tuyệt vời với công thức cho tổng số lần di chuyển: $M(n) = sum(i = 1, n, m(i)) = sum(i = 1, n, 2^{i-1}) = 1 + 2 + ... + 2^{n-1} = 2^n - 1$.

1.3. Câu 3

Để xây dựng thuật toán không đệ quy cho câu đố Tháp Hà Nội, ta có thể dùng stack.

Mô tả thuật toán bằng Python:

```python
def Tower_of_Hanoi(n):
    stack = [(1, 3, n, n)]

    while stack:
        start, end, size, largest = stack.pop()

        if size == 1:
            print("Move disk", largest, "from rod", start, "to rod", end)
        else:
            auxiliary = 6 - start - end
            stack.append((auxiliary, end, size - 1, largest - 1))
            stack.append((start, end, 1, largest))
            stack.append((start, auxiliary, size - 1, largest - 1))
```

Hàm Tower_of_Hanoi nhận đầu vào là một số nguyên n, đại diện cho số đĩa trong câu đố Tháp Hà Nội. Hàm sử dụng một stack để theo dõi các đĩa cần di chuyển. Ban đầu, stack chứa một bộ bốn duy nhất (1, 3, n, n), đại diện cho trạng thái ban đầu của câu đố (tất cả các đĩa đều nằm trên cột đầu tiên và cần được di chuyển đến cột thứ ba).

Vòng lặp while tiếp tục cho đến khi không còn bộ bốn nào trên stack. Trong mỗi lần lặp, hàm pop bộ bốn đầu tiên từ stack và trích xuất các giá trị start (cột bắt đầu), end (cột kết thúc), size (số đĩa trong cột bắt đầu) và largest (đĩa lớn nhất trong các đĩa trong cột bắt đầu). Nếu size là 1, hàm in một thông báo cho biết đĩa largest phải được di chuyển từ cột start đến cột end. Nếu không, hàm đẩy ba bộ bốn mới vào stack theo thứ tự ngược với thứ tự như sau:

1. Một bộ bốn đại diện cho bài toán con di chuyển size - 1 đĩa nhỏ nhất từ cột start đến cột auxiliary, sử dụng cột end làm cột tạm.

2. Một bộ bốn đại diện cho việc di chuyển đĩa largest từ cột start đến cột end.

3. Một bộ bốn đại diện cho bài toán con di chuyển size - 1 đĩa nhỏ nhất từ cột auxiliary đến cột end, sử dụng cột start làm cột tạm.

Thuật toán này hoạt động bằng cách chia bài toán thành các bài toán con

nhỏ hơn và giải quyết chúng một cách tuần tự sử dụng một stack.

2. Bài tập 2: QuickSort

Quicksort is one of the fastest sort-algorithm. Below is the example quicksort code:

```
def QuickSort(arr):
    if len(arr) <= 1:
        return arr
    else:
        pivot = arr[0]
        left = [x for x in arr[1:] if x <= pivot]
        right = [x for x in arr[1:] if x > pivot]
        return QuickSort(left) + [pivot] + QuickSort(right)
```

Set up a recurrence relation, with an appropriate initial condition, for the number of times the basic operation is executed for Quicksort algorithm. And solve it for the best case, worst case and average case, then conclude the time complexity for each case.

Dịch sang tiếng Việt:

Quicksort là một trong những thuật toán sắp xếp nhanh nhất. Dưới đây là code mẫu ccuar quicksort:

```
def QuickSort(arr):
    if len(arr) <= 1:
        return arr
    else:
        pivot = arr[0]
        left = [x for x in arr[1:] if x <= pivot]
        right = [x for x in arr[1:] if x > pivot]
        return QuickSort(left) + [pivot] + QuickSort(right)
```

Thiết lập một quan hệ đệ quy, với điều kiện ban đầu phù hợp, cho số lần thực hiện thao tác cơ bản trong thuật toán Quicksort. Và giải quyết nó cho trường hợp tốt nhất, trường hợp xấu nhất và trường hợp trung bình, sau đó rút ra kết luận về độ phức tạp thời gian cho mỗi trường hợp.

Giải Để thiết lập quan hệ đệ quy cho số lần thực hiện thao tác cơ bản

trong thuật toán Quicksort, chúng ta có thể sử dụng hàm $T(n)$ để biểu thị số lần thực hiện thao tác cơ bản trên một mảng có n phần tử.

Điều kiện ban đầu là $T(1) = 0$, bởi vì một mảng chỉ chứa một phần tử thì không cần phải sắp xếp.

Quan hệ đệ quy có thể được thiết lập như sau:

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

Trong đó k là vị trí chốt được chọn trong mảng đầu vào. Cụ thể, $T(k)$ là số lần thực hiện thao tác cơ bản trên phần tử trái của chốt và $T(n-k-1)$ là số lần thực hiện thao tác cơ bản trên phần tử phải của chốt.

Ta có thể giải quyết quan hệ đệ quy cho các trường hợp khác nhau để tìm độ phức tạp thời gian của thuật toán Quicksort:

1. Trường hợp tốt nhất: Khi mảng luôn được chia ra thành hai phần bằng nhau ở mỗi lần đệ quy. Khi đó:

$$T(n) = 2T(n/2) + n - 1 = 2T(n/2) + \Theta(n)$$

Theo định lý Master, $T(n) = \Theta(nlog_2 n)$

2. Trường hợp xấu nhất: Khi mảng luôn được chia ra thành hai phần , một phần kích thước 0, một phần kích thước n - 1 ở mỗi lần đệ quy. Khi đó:

$$\begin{aligned} T(n) &= T(n-1) + n - 1 \\ &= T(n-2) + n - 1 + n - 2 \\ &= T(1) + (n-1) + (n-2) + ... + 1 \\ &= \frac{n(n-1)}{2} \end{aligned}$$

Vậy $T(n) = \Theta(n^2)$

3. Trường hợp trung bình: Với mọi chốt chặn vị trí i $(0 \leq i < n)$:

- Thời gian để chia mảng: cn

- Một phần chứa i phần tử, phần còn lại chứa $n-i-1$ phần tử, khi đó $T(n) = T(i) + T(n-i-1) + cn$

Vậy thời gian trung bình cho sắp xếp là:

$$\begin{aligned} T(n) &= \frac{1}{n}\sum_{i=0}^{n-1} T(i) + T(n-i-1) + cn \\ &= \frac{2}{n}(T(0) + ... + T(n-1)) + cn \end{aligned}$$

Hay ta có thể viết:

$$nT(n) = 2(T(0) + ... + T(n-1)) + cn^2$$

Từ đó ta có:

$$(n-1)T(n-1) = 2(T(0) + ... + T(n-2)) + c(n-1)^2$$

Khi đó:

$$nT(n) - (n-1)T(n-1) = 2T(n-1) + 2cn - c \approx 2T(n-1) + 2cn$$

Vì vậy $nT(n) \approx (n+1)T(n-1) + 2cn$ hay:

$$\begin{aligned} &\frac{T(n)}{n+1} = \frac{T(n-1)}{n} + \frac{2c}{n+1} \\ \iff &\frac{T(n)}{n+1} - \frac{T(n-1)}{n} = \frac{2c}{n+1} \end{aligned}$$

Ta có:

$$\begin{aligned} &\frac{T(n)}{n+1} + \frac{T(n-1)}{n} + ...\frac{T(1)}{2} \\ -\frac{T(n-1)}{n} &- \frac{T(n-2)}{n-1} - ... - \frac{T(0)}{1} \\ &= \frac{2c}{n+1} + \frac{2c}{n} + ... + \frac{2c}{2} \end{aligned}$$

hay:

$$\frac{T(n)}{n+1} = \frac{T(0)}{1} + 2c(\frac{1}{2} + \frac{1}{3} + ... + \frac{1}{n+1}) \approx 2c(H_{n+1} - 1) \approx c' log_2 n$$

$$\iff T(n) \approx (n+1)c' log_2 n$$

Vậy $T(n) = \Theta(n log_2 n)$

3. Bài tập 3: EXP

a. Design a recursive algorithm for computing 2^n for any nonnegative integer n that is based on the formula $2^n = 2^(n-1) + 2^(n-1)$.

b. Set up a recurrence relation for the number of additions made by the algorithm and solve it.

c. Draw a tree of recursive calls for this algorithm and count the number of calls made by the algorithm.

d. Is it a good algorithm for solving this problem?

Dịch sang tiếng Việt:

a. Hãy thiết kế một thuật toán đệ quy để tính 2^n cho bất kỳ số nguyên không âm n nào dựa trên công thức $2^n = 2^(n-1) + 2^(n-1)$.

b. Thiết lập một quan hệ đệ quy cho số phép cộng được thực hiện bởi thuật toán và giải quyết nó.

c. Vẽ một cây của các lời gọi đệ quy cho thuật toán này và đếm số lượng lời gọi được thực hiện bởi thuật toán.

d. Đó có phải là một thuật toán tốt để giải quyết vấn đề này không?

Giải:

3.1. Câu a

Algorithm 1 Power_of_2(n)

```
if n == 0 then
    return 1
end if
return Power_of_2(n - 1) + Power_of_2(n - 1)
```

3.2. Câu b

Để thiết lập quan hệ đệ quy cho số lần cộng, ta có thể đặt $T(n)$ là số lần cộng để tính 2^n. Theo công thức đệ quy, ta có: $T(n) = 2T(n-1) + 1$

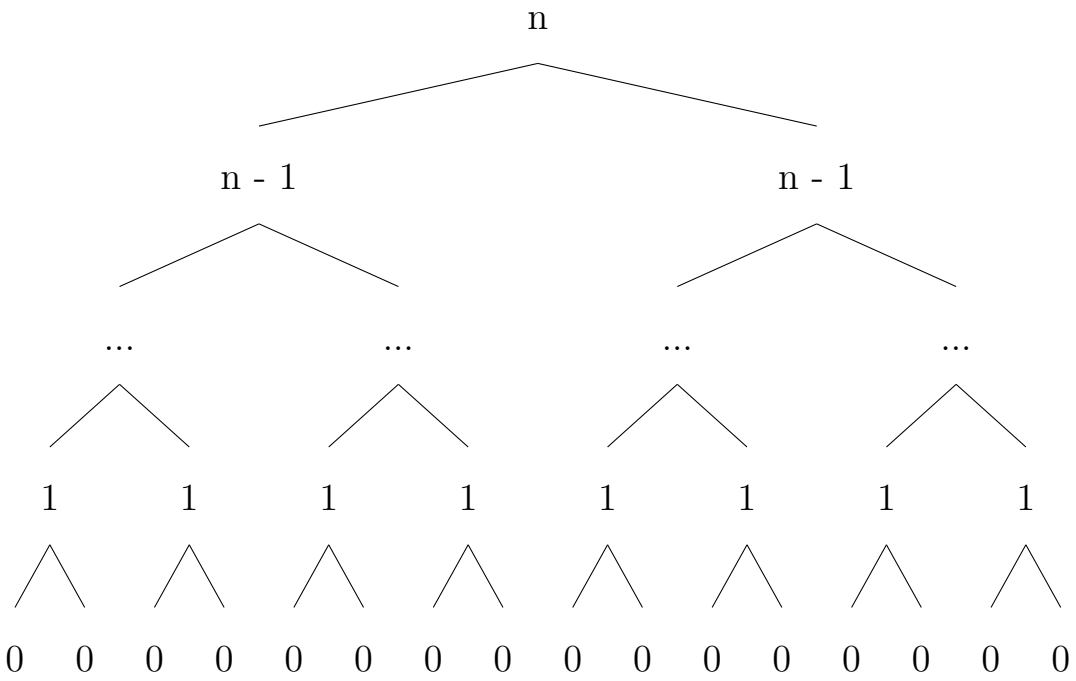
Với trường hợp cơ bản $n = 0$, ta có $T(0) = 0$.

Sử dụng công thức đệ quy trên, ta có thể giải quyết $T(n)$ như sau:

$$\begin{aligned} T(n) &= 2T(n-1) + 1 \\ &= 2^2T(n-2) + 2 + 1 \\ &= 2^3T(n-3) + 2^2 + 2 + 1 \\ &= ... \\ &= 2^nT(0) + 2^{n-1} + 2^{n-2} + ... + 1 \\ &= 2^n - 1 \end{aligned}$$

3.3. Câu c

Cây gọi đệ quy cho thuật toán này như sau:



Từ đó ta thấy hàm được gọi $2^n - 1$ lần.

3.4. Câu d

Đó không thực sự là giải pháp tốt, vì $T(n) = O(2^n)$. Hiện nay, có khá nhiều bài toán rất phức tạp cần phải nhân lũy thừa nhanh. Và đã có thuật toán với độ phức tạp $O(log_2 n)$ để làm điều đó. Ta thấy $log_2 n << 2^n$.