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 đệ quy

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

Mục lục

1 Bài 1: Tower of Hanoi

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.

a. 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.)
Solution:
Gọi thời gian để di chuyển toàn bộ n đĩa từ cột 1 sang cột 3 là $T(n)$. Để thực hiện thao tác này ta cần 3 bước:

- Chuyển $n-1$ đĩa từ cột 1 sang cột 2: tốn thời gian là $T(n-1)$
- Chuyển đĩa cuối cùng (lớn nhất) từ cột 1 sang cột 3: tốn thời gian là $T(1)$
- Chuyển $n-1$ đĩa từ cột 2 sang cột 3: tốn thời gian là $T(n-1)$

Như vậy, tổng thời gian là: $T(n) = 2T(n-1) + T(1)$, với $n > 1$
Theo đề, thời gian di chuyển mỗi chiếc đĩa là 1 phút, tức là $T(1) = 1$ (phút)
Vậy, thời gian cần thiết để di chuyển n chiếc đĩa từ cột 1 sang cột 3:

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

Một cách tổng quan, ta có thể dễ dàng thấy rằng:

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

Từ đó, ta có thể dễ dàng tính được số bước chuyển 64 dĩa là $2^{64} - 1$. Mà giả sử mỗi bước tốn 1 phút. Thì tổng cộng, ta sẽ tốn $2^{64} - 1$ phút 3500 tỷ năm.

b. How many moves are made by the i^{th} largest disk $(1 \leq i \leq n)$ in this algorithm?

Xét mã giả sau:

```
def MoveDisk(n, src, dst, mid):
    if n == 1:
        Move disk from src to dst
        return
    else:
        MoveDisk(n-1, src, mid, dst)
        Move last disk from src to dst
        MoveDisk(n-1, mid, dst, src)
```

Từ mã giả trên, ta dễ dàng thấy rằng, ta chỉ cần di chuyển đĩa lớn nhất một lần, hàm $MoveDisk(n-1)$ được gọi gấp đôi số lần gọi hàm $MoveDisk(n)$. Do đó, số lần di chuyển các đĩa sẽ tăng theo cấp số nhân là 2 theo thứ tự từ lớn đến bé, nghĩa là từ đĩa lớn nhất đến đĩa bé nhất, số lần di chuyển đĩa sẽ là $1, 2, 4, ...2^{n-1}$.

c. Find a nonrecursive algorithm for the Tower of Hanoi puzzle and implement it in the language of your choice

Để giải bài toán này bằng cách sử dụng thuật toán không đệ quy, ta có thể sử dụng *stack* để lưu trữ các bước thực hiện. Cụ thể, thuật toán của ta sẽ như sau:

```
def MoveDist(n):
    # Our stack will contains 3 values: numbers of disk, source,
       destination.
    stack = [(n, 1, 3)]
    while stack is not Null:
        # Get top of stack values
        nDisk, src, dst = stack.pop()
        # If it only has 1 disk left, we just move disk from src to dst.
        if nDisk == 1:
            print("Move disk from {src} to {dst}")
        else:
            # caculate what column left is
            mid = 6 - src - dst
            stack.append((nDisk - 1, src, mid))
            stack.append((1, src, dst))
            stack.append((nDisk - 1, mid, dst))
```

Về phần code cụ thể, nhóm mình sẽ đưa nó vào trong github của nhóm mình (cùng folder với file này).

2 Bài 2: QuickSort

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.
Solution:

- Công thức truy hồi:

$$\begin{aligned} T(n) &= T(k) + T(n-k-1) + O(n) \\ &= T(k) + T(n-k-1) + cn \end{aligned}$$

$T(1) = c$

- Worst case: Khi phần tử chốt luôn là phần tử lớn nhất hoặc nhỏ nhất. Giả sử phần tử chốt là phần tử cuối cùng của mảng đã được sắp xếp tăng dần hoặc giảm dần. Trong quá trình phân chia mảng, 1 mảng con sẽ có $n-1$ phần tử và mảng con còn

lại sẽ rỗng.
Khi đó:

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

- Average case: Khi phẩn từ chốt luôn được chọn ngẫu nhiên.
  Khi đó: $T(n) = O(nlogn)$
- Best case: Khi phần tử chốt luôn là phần tử mang giá trị trung bình của mảng. Trong trường hợp này, kích thước mỗi mảng con gần như cân bằng và mỗi mảng con chứa khoảng $n/2$ phần tử.
  Khi đó:

$$\begin{aligned} T(n) &= T(n/2) + T(n-1-n/2) + cn \\ &= T(n/2) + T(n/2-1) + cn \\ &= 2 \times T(n/2) + cn \\ &= O(nlogn) \end{aligned}$$

3 Bài 3: EXP

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

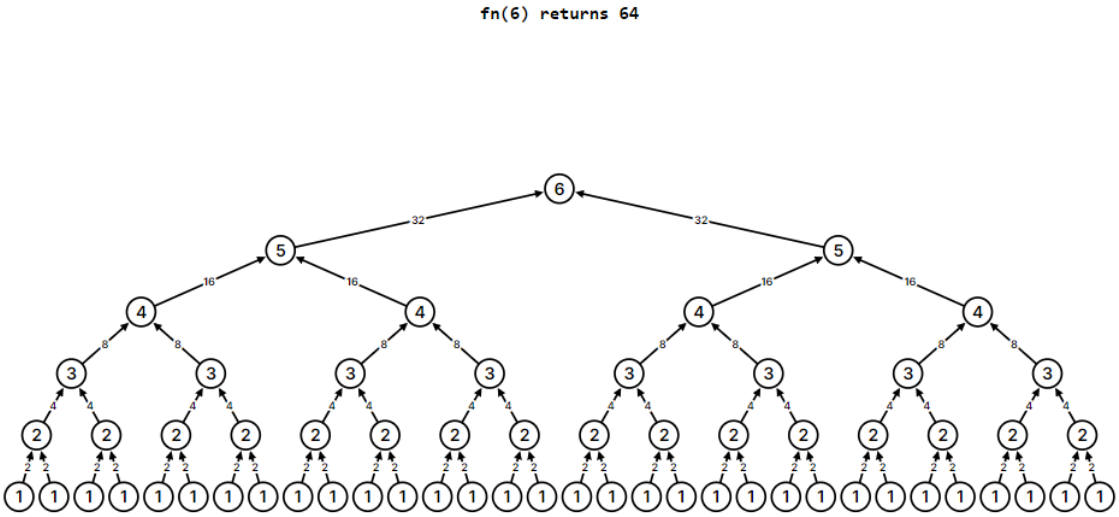
Solution:

- Dựng hàm đệ quy *int Recursion(int n)* với điều kiện dừng là $n = 1$ thì trả về 2.
- Gọi đệ quy tính $Recursion(n-1) + Recursion(n-1)$

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

$$\begin{aligned} T(n) &= 2 \times T(n-1) \\ &= 2 \times 2 \times T(n-2) \\ &= 2 \times \cdots \times 2 \times T(1) \\ &= 2^n \times T(1) \end{aligned}$$

c. Draw a tree of recursive calls for this algorithm and count the number of calls made by the algorithm.
Solution:
Cây đệ quy với $n = 6$



Dễ thấy số lần gọi đệ quy trong thuật toán này là $2^n - 1$.

d. Is it a good algorithm for solving this problem?
Đây không phải là một thuật toán tốt vì với đầu vào là n, hàm đệ quy sẽ bị gọi $2^n - 1$ lần nên độ phức tạp rất lớn.