BÁO CÁO THỰC HÀNH MÔN PHÂN TÍCH THUẬT TOÁN

Lab 3 - Tuần 7

Đinh Anh Huy - 18110103

Bài toán 1. Viết một hàm mà trong đó:

- Input: $f(n)$, hai số nguyên dương a, b chứa khoảng giá trị của n. Ví dụ $a = 10, b = 1000$. Khi đó n chạy từ $10 \to 1000$.
- Output: Chỉ ra độ phức tạp của $f(n) = O(n^\alpha)$, nếu không có dạng này hiện thông báo.

Kiểm tra lại hàm đã có với các trường hợp sau:

(a) $f(n) = n^2$.

(b) $f(n) = n^3 + \cos(n)n^4$.

(c) $f(n) = n^n$.

(d) $f(n) = n^3 + n^2 + n + 1$.

Lời giải

Ý tưởng: Input: $f(n)$, hai số nguyên dương a, b chứa khoảng giá trị của n" nên ta có giá trị của $f(n)$. Do "Output: Chỉ ra độ phức tạp của $f(n) = O(n^\alpha)$" nên $f(n) \sim n^\alpha$. Ta sẽ lấy log cả 2 vế $\log(f(n)) \sim \alpha \log(n)$ thì lúc này ta sẽ xấp xỉ được giá trị của α. Để $f(n)$ có độ phức tạp $O(n^\alpha)$ thì $f(n)$ phải có dạng 1 đa thức bậc α:

$$f(n) = a_0 + a_1 n + a_2 n^2 + ... + a_\alpha n^\alpha$$

Khi đó ta đi xây dựng thuật toán tìm xấp xỉ giá trị α cho mỗi hàm số $f(n)$ với n chạy từ $10 \to 1000$.

```
def complexity(f, a, b, log = False):
    if log:
        log_fn = f
    else:
        log_fn = lambda n: np.log(np.abs(f(n)))
    max = round(log_fn(a)/np.log(a))
    for n in range(a+1, b+1):
        alpha = round(log_fn(n)/np.log(n))
```

```
        if alpha > max:
            max = alpha
    return max
```

Ngoài các tham số f là hàm số cần kiểm tra, a, b là giới hạn của n, thì ta có thêm một tham số là log. Tham số này có chức năng thông báo cho hàm biết rằng hàm f truyền vào có dạng $\log(f(n))$ nếu $log = True$ và ngược lại. Có một số hàm f, ví dụ như $f(n) = n^n$, với n lớn thì hàm tính toán của thư viện numpy không đáp ứng được. Do đó ta cần lấy $log(f)$ để dễ dàng cho việc tính toán hơn.
Ta thu được các giá trị α ứng với từng hàm số $f(n)$ như sau:

```
     Exercise 1
********************
>> For n in range [10, 1000] then
@ alpha for f(n) = n^2:                    2
@ alpha for f(n) = n^3 + cos(n).n^4:       4
@ alpha for f(n) = n^n:                    1000
@ alpha for f(n) = n^3 + n^2 + n + 1:       3
```

Từ kết quả tìm được ở trên, ta thấy rằng các hàm số đều có độ phức tạp là $O(n^\alpha)$ với giá trị α tương ứng với bậc của hàm f.

Bài toán 2. Viết chương trình nhân 2 số nguyên lớn A, B có N chữ số.

- Input: A, B và N.
- Output: $C = A.B$

Bằng 2 phương pháp

(a) Phương pháp truyền thống có độ phức tạp $O(N^2)$.

(b) Phương pháp cải tiến $O(N^{\log 3})$.

Kiểm tra lại chương trình với điều kiện sau:

- Lấy $N = 2^k, k = 10, 11, ..., 32$. A, B là 2 số nguyên có N chữ số lấy ngẫu nhiên. Ứng với mỗi giá trị của N, tính thời gian trung bình để $A.B$.
- So sánh 2 phương pháp để xác định hai phương pháp có độ phức tạp là $O(N^2)$ và $O(N^{\log 3})$.

Lời giải

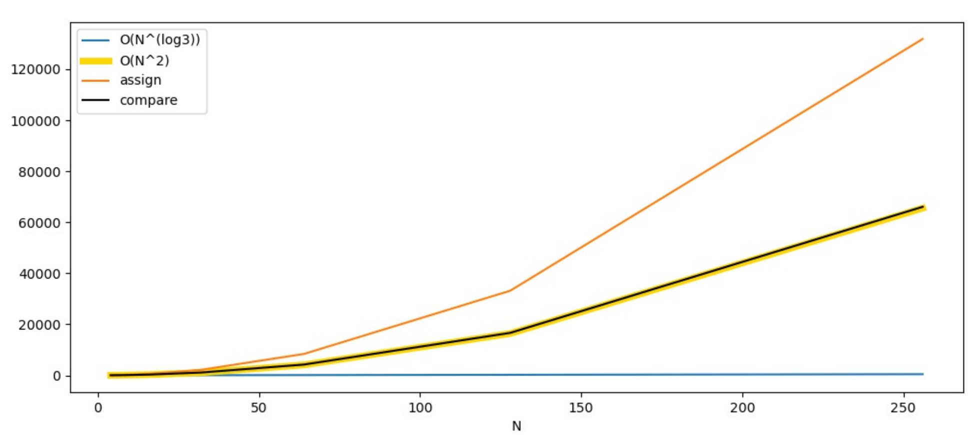
(a) **Phương pháp truyền thống**

Ý tưởng: Với phương pháp truyền thống, ta sẽ xem các số A, B có dạng *string*, Lúc này, ta sẽ từng giá trị của B theo thứ tự từ phải sang trái nhân với A, kết quả sẽ được dịch trái ứng với vị trí $N - i$ tương ứng, với i là vị trí chữ số đang xét của B. Ta dịch trái bằng cách thêm vào chuỗi kết quả tìm được 1 ký tự 0. Kết quả của phép nhân sẽ bằng tổng tất cả các giá trị được tính ở trên sau khi chuyển từ chuỗi sang số.

```
def grade_school_multiply(A, B, N):
    A_str = str(A)
    B_str = str(B)
    result = 0
    for i in range(N-1, -1, -1):
        value = 0
        for j in range(N-1, -1, -1):
            mul1 = str(int(B_str[i])*int(A_str[j])) + '0'*(N-1-j)
            value += int(mul1)

        mul2 = str(value) + '0'*(N-1-i)
        result += int(mul2)
    return result
```

Ta sẽ đi kiểm định về độ phức tạp của phương pháp trên bằng cách vẽ ra đường thời gian tương ứng với mỗi giá trị $N = 2^k, k = 2, ..., 8$.



Từ đồ thị trên, ta có thể thấy rằng các đường biểu thị số phép so sánh và số phép gán của thuật toán trên có nét tương đồng với đường $O(N^2)$, thậm chí đường biểu thị số phép so sánh trùng với đường $O(N^2)$.

(b) **Phương pháp cải tiến (Karatsuba)**

Ý tưởng: Viết A, B dưới dạng

$$A = A_1 * 10^{n/2} + A_2 \quad \text{và} \quad B = B_1 * 10^{n/2} + B_2$$

Đặt $C = A_1 * B_1; D = A_2 * B_2; E = (A_1 + A_2) * (B_1 + B_2) - C - D$. Khi đó

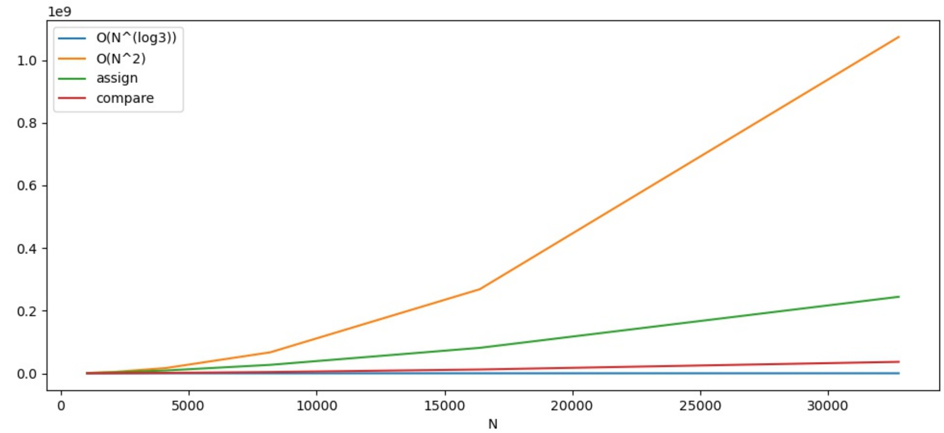
$$A * B = C * 10^n + E * 10^{n/2} + D$$

Ta sẽ lặp lại các bước trên đến khi các số tham gia phép nhân có số phần tử là 1.

```python
def Karatsuba(X, Y, N):
    m = (N//2)
    A1 = X // (10 ** m)
    A2 = X % (10 ** m)
    B1 = Y // (10 ** m)
    B2 = Y % (10 ** m)

    if N == 1:
        return X*Y, assign, compare
    else:
        C = Karatsuba(A1,B1, len(str(A1)))
        D = Karatsuba(A2,B2, len(str(A2)))
        E3 = Karatsuba(A1+A2, B1+B2, len(str(A1+A2)))
        E = E3 - C - D
        result = (10**(m*2))*C + (10**m)*E + D
        return result
```

Ta sẽ đi kiểm định về độ phức tạp của phương pháp cải tiến này với $O(N^{\log 3}$.



Từ độ thị trên ta thấy rằng, các đường biểu thị số lượng phép gán và phép so sánh của thuật toán trên có nét tương đồng với đường $O(N^{\log 3})$ hơn so với đường $O(N^2)$. Điều đó cho thấy rằng, thuật toán trên có độ phức tạp xấp xỉ $O(N^{\log 3})$.

Một ví dụ kiểm định tính chính xác của hai thuật toán trên:

```
     Exercise 2
********************
Example
N =  8
A =  52246461
B =  15119105
>> Implement grade school multiplication:
A*B =  789919729737405
>> Implement karatsuba multiplication:
A*B =  789919729737405
```

Ta thấy rằng kết quả cho ra từ 2 thuật toán hoàn toàn giống nhau. Tuy nhiên do độ phức tạp khác nhau mà thuật toán cải tiến có tốc độ xuất ra kết quả nhanh hơn.