Phân Tích Thuật Toán
Trường Đại Học Khoa Học Tự Nhiên
Nguyễn Thanh Bình

Report lab 5

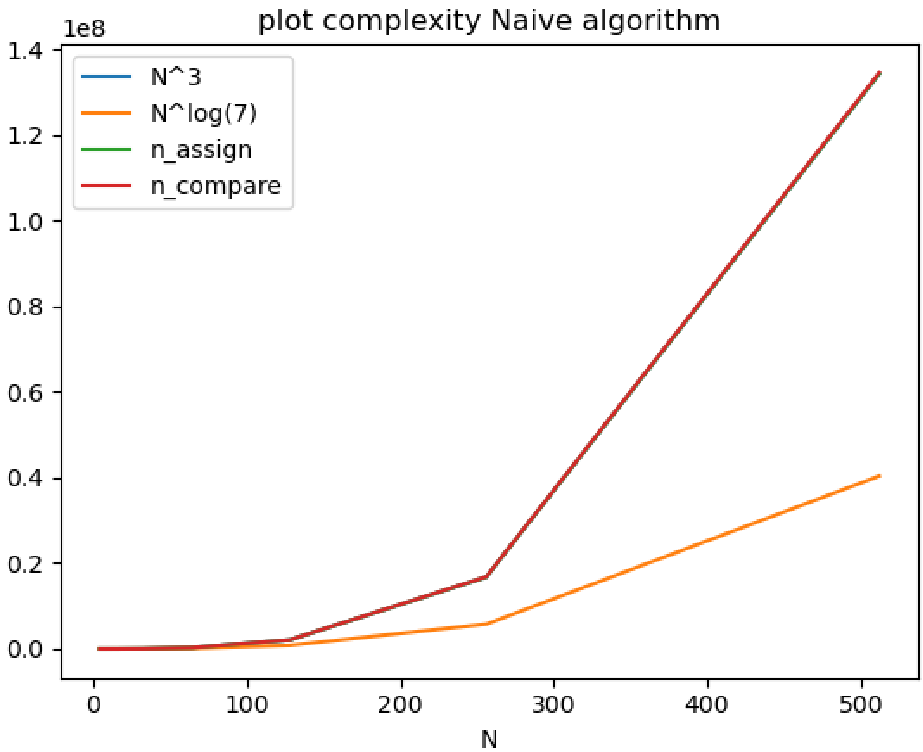
Nguyễn Quốc Bảo - 18110053

Bài 1. Viết chương trình nhân 2 ma trận A và B có kích thước là n $\times$ n với 2 phương pháp như sau:

1. Nhân 2 ma trận như các em đã học trong đại số tuyến tính với độ phức tạp là $O(n^3)$.

```python
def multiply_Matrix(_matrix_A,_matrix_B):
    global counter_assign
    global counter_compare

    # initialization matrix C contain result matrix A multiply matrix B
    C = np.zeros_like(_matrix_A)
    for i in range(len(_matrix_A)):
        counter_compare += 1
        for j in range(len(_matrix_A)):
            counter_compare += 1
            for k in range(len(_matrix_A)):
                C[i,j] += (_matrix_A[i,k]*_matrix_B[k,j])
                counter_assign += 1
                counter_compare += 1
    return C
```

Thuật toán cơ bản cho phép nhân ma trận có thể được thực hiện như một tích chặt chẽ của ba vòng lặp lồng nhau. Đối với mỗi lần lặp của vòng lặp bên ngoài, tổng số lần chạy trong các vòng bên trong sẽ tương đương với độ dài của ma trận. Ở đây, các phép toán số nguyên mất thời gian $O(1)$. Nói chung, nếu độ dài của ma trận là n, thì tổng độ phức tạp về thời gian sẽ là $O(n * n * n) = O(n^3)$


plot complexity Naive algorithm
1e8
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
N^3
N^log(7)
n_assign
n_compare
0
100
200
300
400
500
N

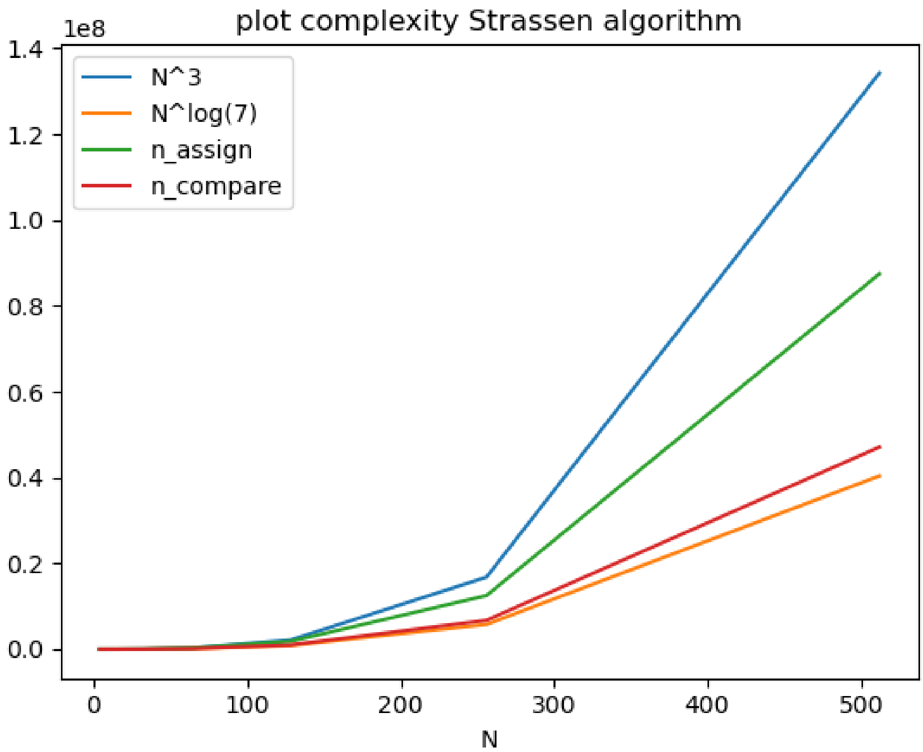
2. Dùng kỹ thuật **Chia để trị** để nhân 2 ma trận trên với độ phức tạp là $O(n^{log7})$.

```python
def strassen(A,B):
    """
        Implementation of the strassen algorithm.
    """
    global counter_assign
    global counter_compare


    counter_compare += 1
    if len(A) == 1:
        return A*B # conventionally computed
    else:
        n_new = len(A) // 2
        counter_assign += 1

        # dividing the matrices in 4 sub-matrices:
        # matrix A into a11 , a12, a21, a22.
        # matrix B into b11 , b12, b21, b22.

        # Calculating p1 to p7:

        # p1 = (a11+a22) * (b11+b22)
        p1 = strassen(A[:n_new, :n_new] + A[n_new:,n_new:], B[:n_new,:n_new] + B[n_new:,n_new:])

        # p2 = (a21+a22) * (b11)
        p2 = strassen(A[n_new:,:n_new] + A[n_new:,n_new:], B[:n_new, :n_new])

        # p3 = (a11) * (b12 - b22)
        p3 = strassen(A[:n_new, :n_new], B[:n_new, n_new:] - B[n_new:,n_new:])

        # p4 = (a22) * (b21 - b11)
        p4 = strassen(A[n_new:,n_new:], B[n_new:,:n_new] - B[:n_new, :n_new] )
        # p5 = (a11+a12) * (b22)
        p5 = strassen(A[:n_new, :n_new] + A[:n_new, n_new:], B[n_new:,n_new:])

        # p6 = (a21-a11) * (b11+b12)
        p6 = strassen(A[n_new:,:n_new] + A[:n_new, :n_new] , B[:n_new, :n_new] + B[:n_new, n_new:])

        # p7 = (a12-a22) * (b21+b22)
        p7 = strassen( A[:n_new, n_new:] - A[n_new:,n_new:], B[n_new:,:n_new] + B[n_new:,n_new:])

        counter_assign += 7

        # calculating c11, c12, c21 and c22:

        c11 = p1 + p4 - p5 + p7  # c11 = p1 + p4 - p5 + p7

        c12 = p3 + p5 # c12 = p3 + p5

        c21 = p2 + p4 # c21 = p2 + p4

        c22 = p1 + p3 - p2 + p6  # c22 = p1 + p3 - p2 + p6

        counter_assign += 4


        # Combining the 4 quadrants into a single matrix by stacking horizontally and vertically.
        C = np.vstack((np.hstack((c11, c12)), np.hstack((c21, c22))))
        counter_assign += 1

        return C
```

Giải pháp này dựa trên đệ quy.
- Trong bước đầu tiên, chúng tôi chia các ma trận đầu vào thành các ma trận con có kích thước $(n/2, n/2)$. Bước này có thời gian thực hiện $O(1)$.
- Ở bước 2, ta tính 10 phép tính cộng / trừ mất thời gian $O(n^2)$.
Trong bước 3, ta thực hiện 7 lần gọi đệ quy để tính toán P1 đến P7. Đầu ra của bước này sẽ là 7 ma trận có chiều là n/2. Bước này cần có thời gian $7T(n/2)$.

- Cuối cùng, bằng cách cộng và trừ các ma trận con P_i, chúng ta nhận được ma trận kết quả C. Sự phức tạp về thời gian của bước này sẽ là $O(n^2)$.
- Do đó, **tổng độ phức tạp** về thời gian của thuật toán này sẽ là:

$$T(n) = 7T(n/2) + O(n^2) = O(n^{log(7)}) = O(n^{2.8074})$$



Ta có kết quả sau khi chạy

```
---------------------------------------- Exercise 1 ----------------------------------------
--------------------------------------------------
>> N =  4
>> A =
 [[ 15  34 980 481]
 [546 876 338 351]
 [344 325 972 655]
 [967 747 629 313]]
>> B =
 [[759 133 927  70]
 [514 464 700  91]
 [488 280 222  53]
 [571 997 726 605]]
----------------By Traditional medthod----------------
C = A.B =
[[ 781752  771728  604471  347089]
 [1230043  923669 1449204  348205]
 [1276487 1121747 1237702  501446]
 [1603586  963400 1786185  358369]]
When N = 4 , then count_compare: 84 and count_assign: 64
----------------By Strassen medthod----------------
C = A.B =
[[ 781752  771728  604471  347089]
 [1230043 2455709 1449204  917115]
 [1276487 1121747 1370834  545276]
 [1603586 3070088 5754225 2886281]]
When N = 4 , then count_compare: 57 and count_assign: 104
```

Kết quả đầy đủ trong tệp **result-ex1.txt**

Bài 2. Có thể nhân 2 ma trận n $\times$ n với độ phức tạp là $O(n^2)$ không? Nếu có thì chứng minh và cài đặt thuật toán. Ngược lại, giải thích nguyên nhân vì sao không thể nhân 2 ma trận có độ phức tạp như vậy

Cách chuẩn để nhân ma trận (m , n) với ma trận (n , p) có độ phức tạp $O(mnp)$. Nếu là ma trận vuông n , thì là $O(n^3)$, không phải $O(n^2)$. Nó sẽ không phải là $O(n^2)$ trong trường hợp chung. Nhưng có những thuật toán nhanh hơn cho các loại ma trận cụ thể. Từ tháng 12 năm 2020, thuật toán nhân ma trận với độ phức tạp tiệm cận tốt nhất chạy trong thời gian $O(n^{2,3728596})$.