Pontificia Universidad Católica de Chile
Departamento de Ciencia de la Computación
IIC2283 - Diseño y Análisis de Algoritmos

Ayudantía 4 - Dividir y Conquistar

Septiembre de 2025

Paula Grune, Manuel Villablanca

Pregunta 1

Existe un algoritmo para multiplicar matrices eficientemente llamado el algoritmo de Strassen. Este define la siguiente formula:

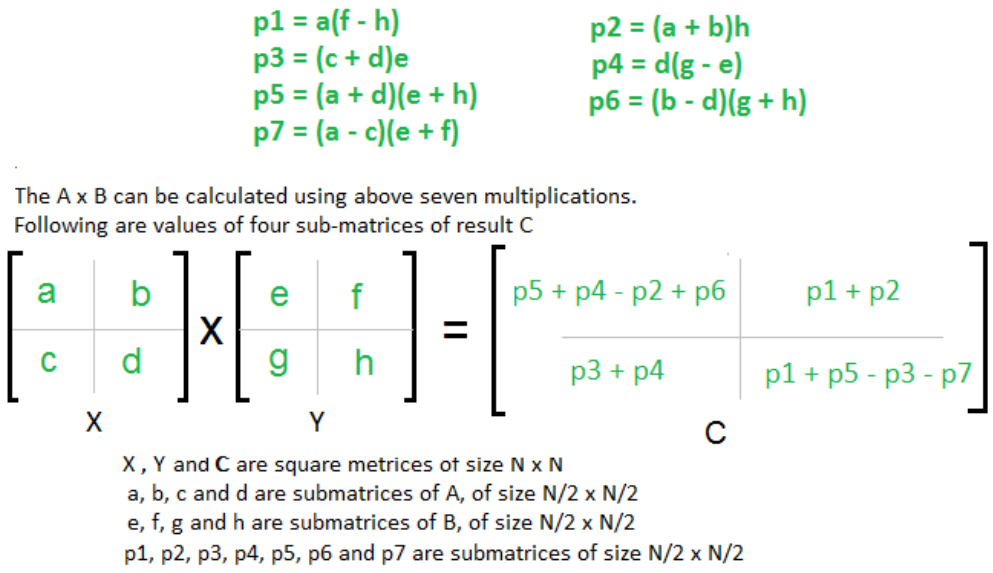


Figura 1: Formula de Strassen

a) Diseñe un algoritmo de dividir y conquistar para multiplicar 2 matrices que utilize la formula de Strassen.

b) Calcule la complejidad del algoritmo dado en el punto anterior.

Solución

```
import numpy as np

def split(matrix):
    """
    Splits a given matrix into quarters.
    Input: nxn matrix
    Output: tuple containing 4 n/2 x n/2 matrices
    corresponding to a, b, c, d
    """
    row, col = matrix.shape
    row2, col2 = row//2, col//2
    return matrix[:row2, :col2], matrix[:row2, col2:],
    matrix[row2:, :col2], matrix[row2:, col2:]

def strassen(x, y):
    """
    Computes matrix product by divide and conquer
    approach recursively.
    Input: nxn matrices x and y
    Output: nxn matrix, product of x and y
    """

    # Base case when size of matrices is 1x1
    if len(x) == 1:
        return x * y

    # Splitting the matrices into quadrants.
    # This will be done recursively
    # until the base case is reached.
    a, b, c, d = split(x)
    e, f, g, h = split(y)

    # Computing the 7 products, recursively (p1, p2...p7)
    p1 = strassen(a, f - h)
    p2 = strassen(a + b, h)
    p3 = strassen(c + d, e)
    p4 = strassen(d, g - e)
    p5 = strassen(a + d, e + h)
    p6 = strassen(b - d, g + h)
    p7 = strassen(a - c, e + f)

    # Computing the values of the 4
    # quadrants of the final matrix c
```

```
    c11 = p5 + p4 - p2 + p6
    c12 = p1 + p2
    c21 = p3 + p4
    c22 = p1 + p5 - p3 - p7

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

    return c
```

$$T(n) = 7 \cdot T\left(\frac{n}{2}\right) + n^2$$

El 7 corresponde a que debemos realizar 7 multiplicaciones de submatrices por cada multiplicación de matrices. Cada cuadrante es de $\frac{n}{2} \times \frac{n}{2}$, por lo que se llama al algoritmo con $n' = \frac{n}{2}$. Y finalmente las operaciones de suma se realizan sobre submatrices de tamaño $\frac{n}{2}$ cada una, por lo que la suma tiene una complejidad de n^2.
Acá también es importante notar que el `vstack` también puede tener a lo más esta misma complejidad, por lo que se deja n^2 como la complejidad de cada iteración.
Usando el teorema maestro, llegamos a una complejidad de:

$$\mathcal{O}\left(n^{\log_2 7}\right)$$

Pregunta 2

Sea $A[1..n]$ un arreglo de enteros positivos que representa a n personas distintas (cada entero representa a una persona). El orden de los elementos del arreglo es arbitrario. Suponga que existe una función $CONOCE(i, j)$ la cual responde `true` si la persona i conoce a la persona j, y `false` en otro caso. El tiempo de ejecución de la función $CONOCE$ es $\mathcal{O}(1)$.

Decimos que una persona i en el arreglo A es una *celebridad* si:

- las $n-1$ personas restantes del arreglo conocen a i
- i no conoce a ninguna de las $n-1$ personas restantes de A

Asuma que un arreglo de tamaño 1 tiene, por defecto, una celebridad. También asume que el tamaño del arreglo siempre es una potencia de 2.

a) Diseñe un algoritmo usando dividir y conquistar que sea lo más eficiente posible.

b) ¿Cuál es la complejidad de su algoritmo?

Solución

Algoritmo CELEBRIDAD(A, i, d):

```
1:  // Caso Base
2:  if d − i + 1 = 1 then
3:     return (true, A[i])
4:  else
5:     // Conquistar
6:     m ← (i + d)/2
7:     (t1, c1) ← CELEBRIDAD(A, i, m)
8:     (t2, c2) ← CELEBRIDAD(A, m + 1, d)
9:     // Combinar
10:    if t1 then
11:       t ← true
12:       for j = m + 1 to d do
13:          if not CONOCE(A[j], c1) or CONOCE(c1, A[j]) then
14:             t ← false
15:          end if
16:       end for
17:       if t then
18:          return (true, c1)
19:       end if
20:    end if
21:    if t2 then
22:       t ← true
23:       for j = i to m do
```

```
            if not CONOCE(A[j], c_2) or CONOCE(c_2, A[j]) then
                t ← false
            end if
        end for
        if t then
            return (true, c_2)
        else
            return (false, 0)
        end if
    else
        return (false, 0)
    end if
end if
```

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + 2n$$

Acá en cada llamada a la función, hacemos 2 llamados recursivos, cada uno de tamaño $\frac{n}{2}$. Y en cada llamada hacemos un trabajo máximo de $2n$, recorriendo el arreglo 1 vez donde para cada indice se llama 2 veces a CONOCE. Dado esto, quedamos con la ecuación de arriba.
Usando teorema maestro, llegamos a la complejidad:

$$\mathcal{O}(n \log n)$$

Pregunta 3

Dado un arreglo $A[1..n]$ de números enteros, se quieren encontrar los índices $1 \leq i \leq j \leq n$ tal que la suma $\sum_{k=i}^{j} A[k]$ tenga el valor máximo posible. Por ejemplo, si

$$A = (3, -4, 5, -2, -2, 6, -3, 5, -3, 2),$$

la respuesta es el subarreglo $A[3..,8]$, con suma máxima 9. Diseñe un algoritmo del tipo Dividir y Conquistar para solucionar este problema. Escriba su solución usando pseudocódigo o algún lenguaje de programación. Asuma que su algoritmo sólo retorna la suma del intervalo correspondiente. Muestre el tiempo de ejecución de su algoritmo.
Hint: El tiempo de ejecución de su algoritmo debe ser el correspondiente a la recurrencia

$$T(n) = \begin{cases} 2T\left(\frac{n}{2}\right) + O(n), & n > 1 \\ 1, & n = 1 \end{cases}$$

Solución: Usando lenguaje C.

```
int maxSubArray(int A[], int izq, int der) {
    if (der - izq + 1 == 1)
        return A[izq];
    else {
        /* Dividir implícito */
        int m = (izq + der + 1) / 2;

        /* Conquistar */
        int s1 = maxSubArray(A, izq, m);
        int s2 = maxSubArray(A, m + 1, der);

        /* Combinar */
        int s = A[m];
        int M1 = A[m];
        for (int j = m - 1; j >= izq; j--) {
            s += A[j];
            if (s > M1) M1 = s;
        }

        s = A[m + 1];
        int M2 = A[m + 1];
        for (int j = m + 2; j <= der; j++) {
            s += A[j];
            if (s > M2) M2 = s;
        }
```

```
        return max(s1, s2, M1 + M2);
    }
}
```

El tiempo de ejecución es:

$$T(n) = 2 \cdot T\left(\tfrac{n}{2}\right) + O(n)$$

De acuerdo al Teorema Maestro, la complejidad es:

$$\mathcal{O}(n \log n)$$