Divide y vencerás

Práctica calificada I

Jarem Villalobos[1] Joaquín Aynaya[2]

Lunes 8 de septiembre de 2025

Moda de un vector

El algoritmo para resolver este problema consiste en dividir el vector en dos partes: derecha e izquierda. Luego, se calculará un mapa (item:frecuencia) de frecuencias de ambos subarrays y se combinarán en un solo mapa usando un procedimiento en $\mathcal{O}(n)$ pasos, obteniendo un algoritmo en $\mathcal{O}(n \log(n))$ Para lograr que la combinación se realize en $\mathcal{O}(n)$, se debe combinar el arreglo más pequeño en el más grande.

Moda de un vector (Pseudocódigo)

```
Procedimiento Moda(A)
    lista_freq = frecuencias(A,
                 0,A.length-1)
    modas = []
    max_freq = Maximo valor
                 en lista_freq
    Para cada (u,v) en lista_freq
        Si (v == max_freq) Entonces
            modas.append(u)
        FinSi
    FinPara
    Devolver modas
FinProcedimiento
---------------------------
Procedimiento combinar(A,B)
    Para cada (u,v) en B
        Si(No existe u en A) Entonces
            A[u] = v
        SiNo
            A[u] += v
        FinSi
    FinPara
FinProcedimiento
```

```
Procedimiento frecuencias(A,inicio,fin)
    Si(inicio<=fin)
        Si(inicio == final)
            Devolver (A[inicio],1)
        FinSi
        mitad = (inicio + fin)/2
        izq = frecuencias(A,inicio,mitad)
        der = frecuencias(A,mitad+1,final)
        Si(left.length > der.length):
            combinar(left,right)
            Devolver left
        SiNo
            combinar(right,left)
            Devolver right
    FinSi
    Devolver ()
FinProcedimiento
```

Moda de un vector (Complejidad)

Tenemos que:

$$F_{Moda}(n) = \mathcal{O}(1)$$
$$F_{Moda}(n) = F_{frecuencias}(n) + \mathcal{O}(n)$$

Además, por cada llamada a *frecuencias* con tamaño n, se llama dos veces a *frecuencias* con tamaño $n/2$ y una vez a *Combinar* con un tamaño $k \leq n/2$, por lo que:

$$F_{frecuencias}(n) = 2F_{frecuencias}(n/2) + \mathcal{O}(n)$$

Aquella fórmula es conocida, por ende $F_{frecuencias} = \mathcal{O}(n \log(n))$. Dado que $O(n) \subset O(n \log(n))$, se concluye que $F_{moda}(n) = \mathcal{O}(n \log(n))$

Multiplicación de números grandes (Karatsuba)

La multiplicación de dos números x e y puede ser resuelta de manera recursiva. Sea n la longitud del número con mayor longitud. Sea $m = n/2$. Podemos representar ambos números como:

$$x = a \times 10^m + b$$
$$y = c \times 10^m + c$$

La multiplicación puede ser computada usando la fórmula:

$$xy = ac \times 10^{2m} + (ad + bc) \times 10^m + bd$$

Sin embargo eso requiere calcular 4 multiplicaciones distintas. Para mejorar la situación podemos hacer

$$(ad + bc) = (a + b)(d + c) - ac - bd$$

Con lo cual, sólo sería necesario calcular 3 multiplicaciones: ac, bd y $(a + b)(c + d)$

Multiplicación de números grandes (Pseudocódigo)

```
Procedimiento Multiplicar(num1,num2)
    Si (num1.length <=1
        AND num2.length <= 1) Entonces
        return num1 * num2
    FinSi

    n = max(num1.length,num2.length)
    m = n/2

    a,b = Partir(num1,m)
    c,d = Partir(num2,m)

    p1 = Multiplicar(a,c)
    p2 = Multiplicar(b,d)
    p3 = Multiplicar(Sumar(a,b),
                     Sumar(c,d))
    p3 = Restar(Restar(p3,p1),p2)
    Devolver Sumar(Sumar(shl(p1,2*m),
                     shl(p3,m)),p2)
FinProcedimiento
```

```
Procedimiento Sumar(num1,num2)
    resultado = ""
    carry = 0
    Si(num2.length
        >num1.length) Entonces
        Intercambiar(num1,num2)
    FinSi
    bias = num1.length - num2.length
    Para i = num1.legnth-1 Hasta 0 Con Pa
        Si((i-bias)>=0) Entonces
            sum = num1[i] +
                num2[i-bias] + carry
        SiNo
            sum = num1[i] + carry
        FinSi
        resultado.insertFirst(sum%10)
        carry = sum/10
    FinPara
    Si(carry == 1) Entonces
        resultado.insertFirst(carry)
    FinSi
    Devolver resultado
FinProcedimiento
```

Multiplicación de números grandes (Pseudocódigo)

```
Procedimiento Restar(num1,num2)
    Si(num2.length
        > num1.length) Entonces
        Intercambiar(num1,num2)
    FinSi
    resultado = ""
    bias = num1.length - num2.length
    carry = 0
    Para i = num1.length-1
        Hasta 0 Con Paso -1
        Si((i-bias)>=0) Entonces
            sub = num1[i] -
            num[2] - carry
        SiNo
            sub = num1[i]-carry
        FinSi
        Si(sub<0) Entonces
            sub += 10
            carry = 1
        SiNo
            carry = 0
        FinSi
        resultado.insertFirst(sub)
    FinPara
    Devolver resultado
FinProcedimiento
```

```
Procedimiento Partir(num,m)
    Si( m >= num1) Entonces
        Devolver ("0",num)
    SiNo
        indice = num.length - m
        Devolver (num[0,...,m-1],
                  num[m,...])
    FinSi
FinProcedimiento
Procedimiento shl(num,m)
    resultado = num
    Para cada i en [0,...,m-1]
        resultado.append("0")
    FinPara
FinProcedimiento
```

Multiplicación de números grandes (Complejidad)

Para analizar el algoritmo, es necesario saber que las operaciones especificadas en el anterior pseudocódigo, como *Sumar*, *Restar*, *shiftLeft*, *shiftRight* son procedimientos que en el peor caso toman $\mathcal{O}(n)$ (Ver implementación)
Entonces tendríamos que la función recursiva de *Multiplicar* en función del número de dígitos sería:

$$F(1) = \mathcal{O}(1)$$
$$F(n) = 3F(n/2) + \mathcal{O}(n)$$

Cuya expresión general termina siendo

$$F(n) = n^{log_2(3)} + cn$$

Para alguna constante c . Así, $F(n) = \mathcal{O}(n^{\log_2(3)}) \approx \mathcal{O}(n^{1.585})$

Multiplicación de matrices (Strassen)

Dadas dos matrices $A^{n\times m}$ y $B^{m\times p}$ expresadas por sus submatrices:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \;,\; B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

Para hallar el producto $AB = C$, se calcula de manera recursiva utilizando sus submatrices usando la expresión:

$$C = \begin{bmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{bmatrix}$$

Sin embargo, eso implica calcular 8 multiplicaciones, lo que haría que no existiera diferencia entre este método y el método tradicional $\mathcal{O}(n^3)$.

Multiplicación de matrices (Strassen)

Se puede definir las matrices:

$$
\begin{aligned}
M_1 &= (A_{11} + A_{22}) \times (B_{11} + B_{22}) & M_2 &= (A_{21} + A_{22}) \times B_11 \\
M_3 &= A_{11} \times (B_{12} - B_{22}) & M_4 &= A_{22} \times (B_{21} - B_{11}) \\
M_5 &= (A_{11} + A_{12}) \times B_{22} & M_6 &= (A_{11} - A_{21}) \times (B_{11} + B_{12}) \\
M_7 &= (A_{12} - A_{22}) \times (B_{21} + B_{22})
\end{aligned}
$$

Tal que ahora:

$$
\begin{aligned}
C_{11} &= M_1 + M_7 + M_4 - M_5 \\
C_{12} &= M_3 + M_5 \\
C_{21} &= M_2 + M_4 \\
C_{22} &= M_1 + M_3 - M_2 - M_6
\end{aligned}
$$

Con lo que ahora sólo se calculan 7 multiplicaciones, en lugar de 8

Multiplicación de matrices (Pseudocódigo)

```
Procedimiento MultiplicarMatrices(A, B):
    filasA <- numero de filas de A
    columnasA <- numero de columnas de A
    filasB <- numero de filas de B
    columnasB <- numero de columnas de B

    Si columnasA != filasB:
        Error: "Matrices incompatibles"
    FinSi

    n <- maximo(filasA, columnasA, filasB, columnasB)
    n2 <- siguientePotenciaDe2(n)

    A_pad <- rellenarConCeros(A, n2, n2)
    B_pad <- rellenarConCeros(B, n2, n2)

    C_pad <- Strassen(A_pad, B_pad)

    C <- submatriz(C_pad, filasA, columnasB)
    devolver C
FinProcedimiento
```

Multiplicación de matrices (Pseudocódigo)

```
Procedimiento Strassen(A, B):
    n <- tamano de A (numero de filas)
    Si n = 1:
        devolver [[A[0][0] * B[0][0]]]
    FinSi

    mid <- n / 2
    A11, A12, A21, A22 <- particionar(A, mid)
    B11, B12, B21, B22 <- particionar(B, mid)

    M1 <- Strassen(A11 + A22, B11 + B22)
    M2 <- Strassen(A21 + A22, B11)
    M3 <- Strassen(A11, B12 - B22)
    M4 <- Strassen(A22, B21 - B11)
    M5 <- Strassen(A11 + A12, B22)
    M6 <- Strassen(A11 - A21, B11 + B12)
    M7 <- Strassen(A12 - A22, B21 + B22)

    C11 <- M1 + M4 - M5 + M7
    C12 <- M3 + M5
    C21 <- M2 + M4
    C22 <- M1 + M3 - M2 - M6

    C <- unir(C11, C12, C21, C22)
    devolver C
FinProcedimiento
```

Multiplicación de matrices (Complejidad)

En cada llamada recursiva de tamaño $n \times n$:

- Se realizan 7 llamadas recursivas con matrices de tamaño $n/2 \times n/2$
- Se realizan un numero constantes de sumas, restas, particiones y uniones cada una con costo $\mathcal{O}(n^2)$

Por lo que la función recursiva del algoritmo estaría dado por:

$$T(1) = \mathcal{O}(1)$$
$$T(n) = 7T(n/2) + \mathcal{O}(n^2)$$

Resolviendo la recurrencia:

$$T(n) = \mathcal{O}(n^{\log_2(7)}) \approx \mathcal{O}(n^{2.81})$$

Lo cual es menor al costo tradicional de $\mathcal{O}(n^3)$.

Subsecuencia de suma máxima

Dado un arreglo A de tamaño n, éste puede dividirse en dos mitades L y R de tamaño $n/2$. El algoritmo buscará la subsecuencia de suma máxima teniendo en cuenta los tres casos:

- La subsecuencia máxima está en el arreglo izquierdo L
- La subsecuencia máxima está en el arreglo derecho R
- La subsecuencia máxima cruza L y R

El algoritmo devuelve como resultado el máximo entre estos casos.

Subsecuencia de suma máxima(Pseudocódigo)

```
Procedimiento MaxSubsecuencia(arr, izquierda, derecha):
    Si izquierda == derecha entonces
        devolver arr[izquierda]
    FinSi

    medio <- (izquierda + derecha) // 2

    max_izq <- MaxSubsecuencia(arr, izquierda, medio)
    max_der <- MaxSubsecuencia(arr, medio+1, derecha)
    max_cruz <- MaxSubsecuenciaCruzada(arr, izquierda, medio, derecha)

    Si max_izq >= max_der y max_izq >= max_cruz:
        devolver max_izq
    SiNo si max_der >= max_izq y max_der >= max_cruz:
        devolver max_der
    SiNo:
        devolver max_cruz
FinProcedimiento
```

Subsecuencia de suma máxima(Pseudocódigo)

```
Procedimiento MaxSubsecuenciaCruzada(arr, izquierda, medio, derecha):
    suma <- 0
    mazIzq <- -inf
    indiceIzq <- medio
    Para i Desde medio Hasta izquierda Paso -1
        suma <- suma + arr[i]
        Si suma > maxIzq
            maxIzq <- suma
            indiceIzq <- i
        FinSi
    FinPara
    suma <- 0
    maxDer <- -inf
    indiceDer <- medio + 1
    Para j Desde medio+1 Hasta derecha
        suma <- suma + arr[j]
        Si suma > maxDer
            maxDer <- suma
            indiceDer <- i
        FinSi
    FinPara
    Devolver maxIzq + maxDer
FinProcedimiento
```

Subsecuencia de suma máxima(Complejidad)

Sea n el tamaño del arreglo, entonces por cada llamada al algoritmo:

- Resuelve 2 subproblemas de tamaño $n/2$
- Calcula la subsecuencia que cruza el medio en tiempo $\mathcal{O}(n)$

Entonces la función recursiva del algoritmo seria:

$$T(1) = \mathcal{O}(1)$$
$$T(n) = 2T(n/2) + \mathcal{O}(n)$$

De lo que se concluye que

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