2.2. Complejidad de las Operaciones Básicas

En esta sección vamos a analizar la complejidad de las operaciones de suma, multiplicación y división con residuo de enteros.

Suma de enteros:

Sean a y b enteros positivos escritos en base β, es decir $a = a_0 + a_1\beta + \cdots + a_{n-1}\beta^{n-1}$ y $b = b_0 + b_1\beta + \cdots + b_{m-1}\beta^{m-1}$. La suma de enteros es término a término, lo cual requiere de a lo más $\max\{n, m\}$ sumas. Adicionalmente se requieren a lo más $\max\{n, m\}$ sumas de lo *que se lleva*. Por lo tanto, el algoritmo de la escuela para sumar enteros utiliza a lo más $2\max\{n, m\}$ sumas. Aquí se supone que el hardware es capaz de realizar la suma de dígitos en base β, es decir, la suma de dos dígitos es un dígito y lo que se *lleva*.

Resumiendo se tiene el siguiente resultado.

Teorema 2.1 La complejidad del algoritmo de la suma de la escuela (en cualquier base) es $O(n)$ para a y b n-dígitos.

Multiplicación de enteros:

Sean $a = a_0 + a_1\beta + \cdots + a_{m-1}\beta^{m-1}$ y $b = b_0 + b_1\beta + \cdots + b_{n-1}\beta^{n-1}$ enteros en base β, con m y n el número de dígitos respectivamente. Si $a \cdot b = c$, entonces c tiene a lo mas $n + m$ dígitos. El algoritmo de la escuela consiste en multiplicar a por cada dígito b_i de b (llevando lo que sea necesario) y luego sumar los resultados obtenidos, teniendo en cuenta las traslaciones necesarias.

Ejemplo 2.12 Consideremos los siguientes enteros en base 6 $a = 8375 = (1, 0, 2, 4, 3, 6)_6$ y $b = 75 = (2, 0, 3)_6$. El algoritmo para hallar el producto de $a \cdot b$ (con multiplicación base 6) es

$$\begin{array}{cccccccc}
 & & 1 & \overset{1}{0} & \overset{2}{2} & \overset{1}{4} & \overset{2}{3} & 5 \\
 & & & & \times & 2 & 0 & 3 \\
\hline
 & & 3 & 1 & 2 & 1 & 5 & 3 \\
 & 0 & 0 & 0 & 0 & 0 & 0 & \\
2 & 0 & 5 & 3 & 1 & 4 & & \\
\hline
2 & 1 & 2 & 4 & 3 & 5 & 5 & 3
\end{array}$$

En este caso se utilizó la siguiente tabla de multiplicación de dígitos en base 6:

```
In[10]:= Table[HoldForm[a*b] == BaseForm[a*b, 6],
           {a, 0, 5}, {b, 0, 5}] // TableForm
```

```
Out[10]=
a b = 0_6   a b = 0_6   a b = 0_6   a b = 0_6   a b = 0_6   a b = 0_6
a b = 0_6   a b = 1_6   a b = 2_6   a b = 3_6   a b = 4_6   a b = 5_6
a b = 0_6   a b = 2_6   a b = 4_6   a b = 10_6  a b = 12_6  a b = 14_6
a b = 0_6   a b = 3_6   a b = 10_6  a b = 13_6  a b = 20_6  a b = 23_6
a b = 0_6   a b = 4_6   a b = 12_6  a b = 20_6  a b = 24_6  a b = 32_6
a b = 0_6   a b = 5_6   a b = 14_6  a b = 23_6  a b = 32_6  a b = 41_6
```

```
In[11]:= IntegerDigits[8375, 6]

Out[11]= {1, 0, 2, 4, 3, 5}

In[12]:= IntegerDigits[75, 6]

Out[12]= {2, 0, 3}

In[13]:= BaseForm[8375*75, 6]

Out[13]= 21243553_6
```

Este algoritmo de multiplicación requiere $O(mn)$ multiplicaciones y a lo más $O(n)$ sumas. Por lo tanto el algoritmo de la escuela tiene complejidad del orden de $O(nm)$.

Un problema de este algoritmo de multiplicación es que hay que guardar *los productos parciales y esto ocupa mucho espacio si los números tienen muchos dígitos. Esto se puede mejorar de la siguiente manera. Para los enteros* $a = a_0+a_1\beta+\cdots+a_{m-1}\beta^{m-1}$ *y* $b = b_0+b_1\beta+\cdots+b_{n-1}\beta^{n-1}$, *definimos la siguiente sucesión*

$$b^{(0)} = 0 \quad y \quad b^{(k)} = b_0 + b_1\beta + \cdots + b_{k-1}\beta^{k-1},$$

para $k = 1, 2, \ldots, n$. *Incluyendo lo que se lleva, se tiene que*

$$ab^{(k)} = c_0^{(k)} + c_1^{(k)}\beta + \cdots + c_{m+k-1}^{(k)}\beta^{m+k-1},$$

con $ab^{(n)} = c$. *Para* $k = 1, \ldots, n$, *tenemos que*

$$ab^{(k+1)} = a(b^{(k)} + b_k\beta^k) = ab^{(k)} + ab_k\beta^k,$$

luego los primeros k *dígitos de* $ab^{(k+1)}$ *no cambian.*

Con el objetivo de hacer un análisis más detallado del algoritmo de la escuela para multiplicar enteros, vamos a presentar este algoritmo de manera recursiva.

Supongamos que los enteros a y b tienen el mismo tamaño en base β. En particular, vamos a suponer que a y b tienen $n = 2^t$ dígitos, de no ser así se puede ampliar la longitud de las entradas hasta llegar a la potencia de dos más cercana (adicionando ceros), es decir que la longitud de los factores es a lo mas el doble y este factor no cambia el análisis de complejidad.

Si $a = a_0 + a_1\beta + \cdots + a_{n-1}\beta^{n-1}$ entonces a se puede escribir como

$$\begin{aligned} a &= a_0 + a_1\beta + \cdots + a_{2^t-1}\beta^{2^t-1} \\ &= (a_0 + a_1\beta + \cdots + a_{2^{t-1}-1}\beta^{2^{t-1}-1}) + \beta^{2^{t-1}}(a_{2^{t-1}} + a_{2^{t-1}+1}\beta + \cdots + a_{2^t-1}\beta^{2^{t-1}-1}) \\ &= A + \tilde{A}\beta^{n/2}, \end{aligned}$$

donde A y $\tilde{A}$ son de longitud $n/2$. Análogamente, se tiene que $b = B + \tilde{B}\beta^{n/2}$, con B y $\tilde{B}$ de longitud $n/2$. Al multiplicar estos dos enteros se tiene que

$$a \cdot b = (A + \tilde{A}\beta^{n/2})(B + \tilde{B}\beta^{n/2}) = AB + (A\tilde{B} + \tilde{A}B)\beta^{n/2} + \tilde{A}\tilde{B}\beta^n.$$

Esta última expresión involucra 4 productos y 1 suma. Sea $T(n)$ el costo computacional de multiplicar dos enteros de longitud $n = 2^t$. Del anterior análisis se tiene que

$$T(n) = \underbrace{4T(n/2)}_{\text{4 multiplicaciones de } n/2 \text{ dígitos}} + \underbrace{Cn}_{\text{1 suma de dígitos de longitud lineal en } n}$$

En este caso C es una constante positiva. Iterando esta igualdad se obtiene que

$$\begin{aligned}
T(n) = T(2^t) &= 4T(2^{t-1}) + C2^t \\
&= 4(4T(2^{t-2}) + C2^{t-1}) + C2^t = 4^2T(2^{t-2}) + C2^t(1+2) \\
&= 4^2(4T(2^{t-3}) + C2^{t-2}) + C2^t(2+1) = 4^3T(2^{t-3}) + C2^t(2^2+2+1) \\
&\vdots \\
&= 4^tT(1) + 2^tC(2^{t-1} + 2^{t-2} + \cdots + 1) \\
&= (2^t)^2(T(1) + C - 1/2^t) \\
&= n^2(T(1) + C - 1/n) = O(n^2).
\end{aligned}$$

Resumiendo tenemos el siguiente resultado.

Teorema 2.2 La complejidad del algoritmo de la multiplicación de la escuela (en cualquier base) es $O(n^2)$ para a y b n-dígitos.

2.2.1. Algoritmo de Karatsuba

En 1963 Anatoli Karatsuba encontró una mejora al algoritmo de la escuela para multiplicar enteros. En particular sólo se necesitan 3 multiplicaciones y algunas sumas adicionales.

Sean a y b enteros en base β y de tamaño $n = 2^t$. Es decir que a y b se pueden escribir como $a = A + \tilde{A}\beta^{n/2}$ y $b = B + \tilde{B}\beta^{n/2}$ con $A, \tilde{A}, B, \tilde{B}$ $n/2$ dígitos. Karatsuba observó que el producto $a \cdot b$ se puede escribir como

$$a \cdot b = AB + (\tilde{A}\tilde{B} + AB + (\tilde{A} - A)(B - \tilde{B})\beta^{n/2} + \tilde{A}\tilde{B}\beta^n.$$

Esta última igualdad utiliza 3 productos y 4 sumas-restas.

Teorema 2.3 La complejidad del algoritmo de Karatsuba para multiplicar enteros es $O(n^{\log_2 3}) = O(n^{1.59})$, para una pareja a y b de n-dígitos.

Demostración. Sea $T(n)$ el costo de complejidad del producto de dos n-dígitos, con $n = 2^t$. Teniendo en cuenta que el algoritmo de Karatsuba utiliza 3 productos y 4 sumas-restas, entonces se obtiene que $T(n) = 3T(n/2) + Cn$, con C una constante positiva. Iterando

esta última igualdad se obtiene que

$$\begin{aligned}
T(n) &= 3T(n/2) + Cn = 3T(2^{t-1}) + 2^t C \\
&= 3(3T(2^{t-2}) + 2^{t-1}C) + 2^t C = 3^2 T(2^{t-2}) + 3 \cdot 2^{t-1}C + 2^t C \\
&= 3^2(3T(2^{t-3} + 2^{t-2}C) + 2^t C(3/2+1) = 3^3 T(2^{t-3}) + 2^t C((3/2)^2 + (3/2) + 1) \\
&\vdots \\
&= 3^t T(1) + 2^t C\left(\frac{(3/2)^t - 1}{3/2 - 1}\right) = 3^t T(1) + 2^{t+1}C((3/2)^t - 1) \\
&= 3^t\left(T(1) + 2C - \frac{2^{t+1}}{3^t}C\right) = 2^{\log_2 3^t}\left(T(1) + 2C - \frac{2n}{2^{\log_2 3^t}}C\right) \\
&= n^{\log_2 3}\left(T(1) + 2cC\frac{2n}{n^{\log_2 3}}C\right).
\end{aligned}$$

De esto último se deriva que la complejidad es $O(n^{\log_2 3}) = O(n^{1.59})$. □

La constante C obtenida en la anterior demostración es más grande que la constante en el método de la escuela. Algunos autores han verificado que el algoritmo de Karatsuba es mejor que el algoritmo de la escuela para enteros de al menos 500 dígitos. Este algoritmo se puede extender fácilmente a polinomios.

Ejemplo 2.13 Tomemos los enteros $a = 2925, b = 6872$. Entonces $A = 25, \tilde{A} = 29, B = 72, \tilde{B} = 68$. En este caso se tiene que

$$\begin{aligned}
A \cdot B &= 25 \cdot 72 = 1800 & \tilde{A} \cdot \tilde{B} &= 29 \cdot 68 = 1972 \\
\tilde{A} - A &= 29 - 25 = 4 & B - \tilde{B} &= 72 - 68 = 4
\end{aligned}$$

Luego $\tilde{A}\tilde{B} + AB + (\tilde{A} - A)(B - \tilde{B}) = 1972 + 1800 + 16 = 3788$. Por lo tanto

$$\begin{aligned}
a \cdot b &= 1972(10)^4 + 3788(10)^2 + 1800 \\
&= 19720000 + 378800 + 1800 \\
&= 20100600.
\end{aligned}$$

La identidad en la que se basa el Algoritmo de Karatsuba se puede obtener y generalizar teniendo en cuenta lo siguiente. Sean a y b enteros en base β y de tamaño $n = 2^t$. Es decir que a y b se pueden escribir como $a = A + \tilde{A}X := a(X)$ y $b = B + \tilde{B}X := b(x)$ con $A, \tilde{A}, B, \tilde{B}$ $n/2$ dígitos y $X = \beta^{n/2}$. El producto $c = ab$ se puede pensar como un polinomio de grado 2 en la indeterminada X, $c = ab = C_0 + C_1 X + C_2 X^2 := c(X)$, donde $C_0 = AB, C_1 = A\tilde{B} + \tilde{A}B$ y $C_2 = \tilde{A}\tilde{B}$. Los coeficientes C_0, C_1 y C_2 se pueden determinar evaluando $c(X)$ en tres valores diferentes. Consideremos $X = 0, -1, \infty$. Para un polinomio cualquiera $p(x)$, la notación $p(\infty)$ corresponde al coeficiente principal de p. Por lo tanto,

tenemos que

$$\begin{array}{lllllll} c(\infty) & = & C_2 & = & a(\infty)b(\infty) & = & \tilde{A}\tilde{B}, \\ c(0) & = & C_0 & = & a(0)b(0) & = & AB, \\ c(-1) & = & C_0 - C_1 + C_2 & = & a(-1)b(-1) & = & (A-\tilde{A})(B-\tilde{B}), \end{array}$$

Resolviendo este sistema para C_0, C_1, C_2 se obtienen los coeficientes del algoritmo de Karatsuba, es decir, $C_0 = AB$, $C_2 = \tilde{A}\tilde{B}$ y

$$C_1 = C_0 + C_2 - (A-\tilde{A})(B-\tilde{B}) = AB + \tilde{A}\tilde{B} - (A-\tilde{A})(B-\tilde{B}).$$

El algoritmo de Karatsuba muestra como con algunos cambios en los algoritmos clásicos se pueden obtener mejoras en la complejidad. En el siguiente ejemplo mostramos un algoritmo que mejora la complejidad del producto de matrices que sabemos que es $O(n^3)$, donde n es el tamaño de las matrices.

Ejemplo 2.14 (Algoritmo de Strassen) En este ejemplo vamos a analizar la multiplicación de matrices cuadradas. El algoritmo del curso de álgebra lineal para multiplicar dos matrices cuadradas de tamaño $n \times n$ requiere n multiplicaciones y $n-1$ sumas al hacer el producto punto de una fila de la primera matriz con una columna de la segunda matriz. Este procedimiento se hace n veces para una misma fila y se repite con cada una de las n filas de la primera matriz. Por lo tanto, se hacen $n^2(n+(n-1))$ operaciones, es decir que el algoritmo es cúbico ($O(n^3)$).

Volker Strassen publicó hacia finales de los sesentas un algoritmo que mejora ligeramente el algoritmo clásico para multiplicar matrices. Desde entonces se vienen haciendo mejoras de este. En Octubre de 2020 Josh Alman y Virginia Vassilevska demostraron que el producto de matrices se puede mejorar al orden $O(n^{2.37286})$.

A continuación describimos el algoritmo de Strassen. Considere dos matrices A y B de tamaño $n \times n$ (podemos suponer que n es una potencia de 2, de lo contrario agregamos tantas filas y columnas de ceros como sea necesario para obtener una matriz de tamaño una potencia de 2). Cada una de estas matrices se puede dividir en 4 bloques de tamaño $n/2 \times n/2$:

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} \quad B = \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix}.$$

Es claro que

$$C = A \cdot B = \begin{bmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{bmatrix},$$

donde

$$\begin{aligned}
C_{1,1} &= A_{1,1}B_{1,1} + A_{1,2}B_{2,1},\\
C_{1,2} &= A_{1,1}B_{1,2} + A_{1,2}B_{2,2},\\
C_{2,1} &= A_{2,1}B_{1,1} + A_{2,2}B_{2,1},\\
C_{2,2} &= A_{2,1}B_{1,2} + A_{2,2}B_{2,2}.
\end{aligned}$$

Esta subdivisión es de orden $O(1)$. Se definen las siguientes 10 submatrices S_i, $i = 1, \ldots, 10$

$$\begin{aligned}
&S_1 = B_{1,2} - B_{2,2}, \quad S_2 = A_{1,1} + A_{1,2}, \quad S_3 = A_{2,1} + A_{2,2}, \quad S_4 = B_{2,1} - B_{1,1},\\
&S_5 = A_{1,1} + A_{2,2}, \quad S_6 = B_{1,1} + B_{2,2}, \quad S_7 = A_{1,2} - A_{2,2}, \quad S_8 = B_{2,1} + B_{2,2},\\
&S_9 = A_{1,1} - A_{2,1}, \quad S_{10} = B_{1,1} + B_{1,2}.
\end{aligned}$$

El número de sumas o restas de matrices de tamaño $n/2 \times n/2$ es de orden $O(n^2)$ ya que la adición o resta de matrices es componente a componente. Estas operaciones se hacen en un tiempo constante. En el siguiente paso calculamos las siguientes 7 matrices, lo cual requiere de 7 productos:

$$\begin{aligned}
P_1 &= A_{1,1}S_1 = A_{1,1}B_{1,2} - A_{1,1}B_{2,2},\\
P_2 &= S_2B_{2,2} = A_{1,1}B_{2,2} + A_{1,2}B_{2,2},\\
P_3 &= S_3B_{1,1} = A_{2,1}B_{1,1} + A_{2,2}B_{1,1},\\
P_4 &= A_{2,2}S_4 = A_{2,2}B_{2,1} - A_{2,2}B_{1,1},\\
P_5 &= S_5S_6 = A_{1,1}B_{1,1} + A_{1,1}B_{2,2} + A_{2,2}B_{1,1} + A_{2,2}B_{2,2},\\
P_6 &= S_7S_8 = A_{1,2}B_{2,1} + A_{1,2}B_{2,2} - A_{2,2}B_{2,1} - A_{2,2}B_{2,2},\\
P_7 &= S_9S_{10} = A_{1,1}B_{1,1} + A_{1,1}B_{1,2} - A_{2,1}B_{1,1} - A_{2,1}B_{1,2}.
\end{aligned}$$

Por último, calculamos los bloques $C_{1,1}, C_{1,2}, C_{2,1}$ y $C_{2,2}$ de la siguiente forma:

$$\begin{aligned}
C_{1,1} &= P_5 + P_4 - P_2 + P_6,\\
C_{1,2} &= P_1 + P_2,\\
C_{2,1} &= P_3 + P_4,\\
C_{2,2} &= P_5 + P_1 - P_3 - P_7.
\end{aligned}$$

Por lo tanto, se tiene que si $T(n)$ es el costo computacional de multiplicar matrices de tamaño $n \times n$ (con n una potencia de 2, en particular $n = 2^t$), entonces para $n > 1$ tenemos que $T(n) = 7T(n/2) + O(n^2)$ y $T(1) = O(1) = k$. A partir de esta recurrencia se puede demostrar que $T(n) = O(n^{\log_2 7})$. En efecto, para $n = 2^t$ se tiene que

$$T(n) = \underbrace{7T(n/2)}_{\text{7 multiplicaciones de matrices de tamaño } n/2 \times n/2} + \underbrace{kn^2}_{n^2 \text{ sumas}}$$

Iterando se obtiene que

$$\begin{aligned} T(n) = T(2^t) &= 7T(2^{t-1}) + k2^{2t} \\ &= 7(7T(2^{t-2}) + k2^{2t-2}) + k2^{2t} = 7^2T(2^{t-2}) + k(2^{2t-2} + 2^{2t}) \\ &\vdots \\ &= 7^tT(1) + k(1 + \cdots + 2^{2t-2} + 2^{2t}) \\ &= 7^tT(1) + k\frac{2^{2(t+1)} - 1}{2^2 - 1} \\ &= n^{\log_2 7}T(1) + k\frac{4n^2 - 1}{3}. \end{aligned}$$

Observe que $7^t = 2^{\log_2 7^t} = n^{\log_2 7} = n^{2.80735}$, así que

$$T(n) = n^{2.80735}T(1) + O(n^2),$$

luego $T(n) = O(n^{2.80735})$.

Cerramos esta sección enunciando parte del Teorema Maestro que permite resolver recurrencias de la forma $T(n) = aT(n/b) + f(n)$, donde $a \geq 1$ y $b > 1$ son constantes y $f(n)$ es una función asintótica positiva. Su demostración se puede encontrar en [1].

Teorema 2.4 (Teorema Maestro) Sean $a \geq 1$ y $b > 1$ constantes y $f(n)$ es una función asintótica positiva. Sea $T(n)$ una función definida sobre los enteros no negativos por la recurrencia

$$T(n) = aT(n/b) + f(n),$$

donde n/b se puede interpretar como $\lfloor n/2 \rfloor$ o $\lceil n/2 \rceil$. Entonces $T(n)$ tiene el siguiente comportamiento asintótico

1. Si $f(n) = O(n^{\log_b a - \epsilon})$ para alguna constante $\epsilon > 0$, entonces $T(n) = O(n^{\log_b a})$.
2. Si $f(n) = O(n^{\log_b a})$ entonces $T(n) = O(n^{\log_b a} \ln n)$.

Ejemplo 2.15 En el algoritmo de Strassen se encontró la recurrencia $T(n) = 7T(n/2) + O(n^2)$. Aplicando el Teorema Maestro con $a = 7$ y $b = 2$ se tiene que $f(n) = O(n^2) = O(n^{\log_2 7 - \epsilon})$ con $\epsilon = \log_2 7 - 2 = 2.80735 - 2 = 0.80735$. Por la parte 1 del Teorema Maestro concluimos que $T(n) = O(n^{\log_2 7}) = O(n^{2.80735})$.
En el algoritmo de Karatsuba se encontró la recurrencia $T(n) = 3T(n/2) + O(n)$. Aplicando el Teorema Maestro con $a = 3$ y $b = 2$ se tiene que $f(n) = O(n) = O(n^{\log_2 3 - \epsilon})$ con $\epsilon = \log_2 3 - 1 = 1.58496 - 1 = 0.58496$. Por la parte 1 del Teorema Maestro concluimos que $T(n) = O(n^{\log_2 3}) = O(n^{1.58496})$.