Técnicas Fundamentales

Algoritmo de Karatsuba y Programación Dinámica

Segundo semestre 2022

IIC2283

Prof. Nicolás Van Sint Jan

Recordatorio: Dividir para conquistar

Esta es la forma genérica de una algoritmo que utiliza la técnica de **dividir para conquistar**:

Bosquejo de general de algoritmo

DividirParaConquistar(w)
    if $|w| \leq k$ **then return InstanciasPequeñas**(w)
    else
        Dividir w en $w_1, \ldots, w_\ell$
        for $i := 1$ **to** ℓ **do**
            $S_i :=$ **DividirParaConquistar**(w_i)
        return Combinar($S_1, \ldots, S_\ell$)

¿Cuál es la complejidad de un algoritmo de **dividir para conquistar**?

Recordatorio: Suma de números enteros

Sean $a, b \in \mathbb{Z}$ con $n \geq 1$ dígitos cada uno. Sea

$$c = a + b.$$

Considere **el algoritmo usual de la suma** para calcular c.

Consideramos la **suma de dos dígitos**, **comparación de dos dígitos** y **resta de un número con a lo más dos dígitos con uno de un dígito** como las operaciones a contar, cada una con costo 1.

Preguntas

1. ¿Cuántas operaciones realiza el algoritmo en el peor caso?
2. ¿Cuántos dígitos puede tener c?

¿Se puede **sumar** más rápido que $\mathcal{O}(n)$?

Recordatorio: Multiplicación de números enteros

Sean $a, b \in \mathbb{Z}$ con $n \geq 1$ dígitos cada uno. Sea

$$d = a \cdot b$$

Considere **el algoritmo usual de la multiplicación** para calcular d.

Esta vez tome **la suma** y **la multiplicación de dígitos** como las operaciones a contar, ambas con costo 1.

Preguntas

1. ¿Cuántas operaciones realiza el algoritmo en este caso?
2. ¿Cuántos dígitos puede tener d?

¿Se puede **multiplicar** más rápido que $\mathcal{O}(n^2)$?

Outline

Dividir para conquistar: Multiplicar

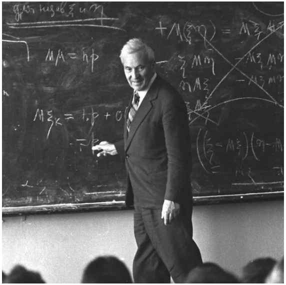
Programación dinámica: Grafos

Outline

Dividir para conquistar: Multiplicar

Programación dinámica: Grafos

Algoritmo de multiplicación de Karatsuba



Andréi Kolmogorov

Anatoli Karatsuba

Algoritmo de multiplicación de Karatsuba

Sean $a, b \in \mathbb{Z}$ con n dígitos cada uno, donde $n = 2^k$ para algún $k \in \mathbb{N}$.

Se puede representar a y b de la siguiente forma:

$$\begin{aligned} a &= a_1 \cdot 10^{\frac{n}{2}} + a_2 \\ b &= b_1 \cdot 10^{\frac{n}{2}} + b_2 \end{aligned}$$

Tenemos entonces que:

$$a \cdot b = a_1 \cdot b_1 \cdot 10^n + (a_1 \cdot b_2 + a_2 \cdot b_1) \cdot 10^{\frac{n}{2}} + a_2 \cdot b_2$$

Algoritmo de multiplicación de Karatsuba

$$a \cdot b = a_1 \cdot b_1 \cdot 10^n + (a_1 \cdot b_2 + a_2 \cdot b_1) \cdot 10^{\frac{n}{2}} + a_2 \cdot b_2$$

Para calcular $a \cdot b$ entonces debemos calcular las siguientes multiplicaciones:

1. $a_1 \cdot b_1$
2. $a_1 \cdot b_2$
3. $a_2 \cdot b_1$
4. $a_2 \cdot b_2$

Obtenemos entonces un algoritmo recursivo

- Para resolver el caso de largo n realizamos 4 llamadas para los casos de largo $\frac{n}{2}$

¿Cuál el la complejidad de este algoritmo?

R: Se puede usar el teorema Maestro para deducir que este algoritmo es de orden $\Theta(n^2)$.

La idea clave en el algoritmo de Karatsuba

Podemos calcular $a \cdot b$ realizando las siguientes multiplicaciones:

1. $c_1 = a_1 \cdot b_1$
2. $c_2 = a_2 \cdot b_2$
3. $c_3 = (a_1 + a_2) \cdot (b_1 + b_2)$

Tenemos entonces que:

$$\boxed{a \cdot b \; = \; c_1 \cdot 10^n + (c_3 - (c_1 + c_2)) \cdot 10^{\frac{n}{2}} + c_2}$$

Esta expresión se conoce como **el algoritmo de Karatsuba**.

¿Cuántas **operaciones** realiza este algoritmo?

Tiempo de ejecución del algoritmo de Karatsuba

Sea $T(n)$ el número de operaciones realizadas en el **peor caso** por el algoritmo de Karatsuba para dos números de entrada con n dígitos cada uno.

Para determinar el orden de $T(n)$ utilizamos la siguiente **ecuación de recurrencia** (con $e \in \mathbb{N}$ una constante):

$$T(n) \;=\; \begin{cases} 1 & n = 1 \\ 3 \cdot T\left(\frac{n}{2}\right) + e \cdot n & n > 1 \end{cases}$$

Tiempo de ejecución del algoritmo de Karatsuba

$$a \cdot b \;=\; c_1 \cdot 10^n + (c_3 - (c_1 + c_2)) \cdot 10^{\frac{n}{2}} + c_2$$

$$T(n) \;=\; \begin{cases} 1 & n = 1 \\ 3 \cdot T\left(\frac{n}{2}\right) + e \cdot n & n > 1 \end{cases}$$

Preguntas

1. ¿Qué supuestos realizamos al formular esta ecuación?
   - n es una potencia de 2 y que $(a_1 + a_2)$ y $(b_1 + b_2)$ tienen $\frac{n}{2}$ dígitos cada uno.
2. ¿Qué representa la constante e?
   - Calcular $(a_1 + a_2)$, $(b_1 + b_2)$, $(c_1 + c_2)$ y $(c_3 - (c_1 + c_2))$.
   - Construir $a \cdot b$ a partir de c_1, c_2 y $(c_3 - (c_1 + c_2))$, lo cual puede tomar tiempo lineal en el peor caso. ¿Por qué?

Resolviendo la ecuación de recurrencia

Utilizando el **Teorema Maestro** obtenemos que $T(n)$ es $\Theta(n^{\log_2(3)})$.

- Pero este resultado es válido bajo los supuestos realizados anteriormente.

¿Cómo debe formularse el algoritmo de Karatsuba en el **caso general**?

Caso general del algoritmo de Karatsuba

En el caso general, representamos las entradas a y b de la siguiente forma:

$$\begin{aligned} a &= a_1 \cdot 10^{\lfloor \frac{n}{2} \rfloor} + a_2 \\ b &= b_1 \cdot 10^{\lfloor \frac{n}{2} \rfloor} + b_2 \end{aligned}$$

La siguiente ecuación de recurrencia para $T(n)$ captura la cantidad de operaciones realizadas por el algoritmo (para constantes e_1, e_2):

$$T(n) = \begin{cases} e_1 & n \leq 3 \\ T\left(\lfloor \frac{n}{2} \rfloor\right) + T\left(\lceil \frac{n}{2} \rceil\right) + T\left(\lceil \frac{n}{2} \rceil + 1\right) + e_2 \cdot n & n > 3 \end{cases}$$

Ejercicio

Demuestre usando inducción constructiva que $T(n)$ es $\mathcal{O}(n^{\log_2(3)})$

- En particular, demuestre lo siguiente:
  $\exists c \in \mathbb{R}^+.\ \exists d \in \mathbb{R}^+.\ \exists n_0 \in \mathbb{N}.\ \forall n \geq n_0.\ T(n) \leq c \cdot n^{\log_2(3)} - d \cdot n$

Outline

Programación dinámica: un primer ingrediente

Al igual que dividir para conquistar, la técnica de **programación dinámica** resuelve un problema dividiéndolo en sub-problemas más pequeños.

Pero a diferencia de dividir para conquistar, en este caso se espera que **los sub-problemas estén traslapados**.

De esta forma se reduce el número de sub-problemas a resolver, de hecho se espera que este número sea pequeño (al menos polinomial).

Contando el número de caminos en un grafo

Sea $G = (V, E)$ un **grafo dirigido**.

Recordar que una secuencia v_1, ..., v_ℓ de elementos en N es un **camino** en G si:

1. $\ell \geq 2$
2. $(v_i, v_{i+1}) \in E$ para cada $i \in \{1, \ldots, \ell - 1\}$

Decimos que un camino v_1, ..., v_ℓ va desde v_1 a v_ℓ, y definimos su largo como (ℓ - 1), vale decir, el número de aristas en el camino.

Ejemplo

$$\begin{tikzcd}
a \arrow[dr] & b \arrow[l] & e \arrow[l] \arrow[d] \\
& d \arrow[ur] & f \arrow[l]
\end{tikzcd}$$

b, a, d, e, f

Contando el número de caminos en un grafo

Dado un grafo $G = (V, E)$, un par de nodos v_i, v_f en V y un número ℓ, queremos desarrollar un algoritmo que cuente el **número de caminos** desde v_i a v_f en G cuyo largo es igual a ℓ

Suponemos que $V = \{1, \ldots, n\}$, $1 \leq \ell \leq n$ y representamos G a través de su **matriz de adyacencia** M tal que:

Si $(i, j) \in E$, entonces $M[i, j] = 1$, en caso contrario $M[i, j] = 0$.

Ejemplo

$$
\begin{tikzcd}
1 \arrow[dr] & 2 \arrow[l] & 4 \arrow[l] \arrow[d] \\
 & 3 \arrow[ur] & 5 \arrow[l]
\end{tikzcd}
\qquad
M = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}
$$

Una primera definición de **ContarCaminos**

Queremos entonces definir la función **ContarCaminos**(M, v_i, v_f, ℓ).

ContarCaminos($M[1 \ldots n][1 \ldots n]$, v_i, v_f, ℓ)
    if $\ell = 1$ **then return** $M[v_i, v_f]$
    else
        $aux := 0$
        for $v_j := 1$ **to** n **do**
            $aux \mathrel{+}= M[v_i, v_j] \cdot$ **ContarCaminos**(M, v_j, v_f, $\ell - 1$)
        return aux

Observe que usamos la notación $C[1 \ldots m][1 \ldots n]$ para indicar que la matriz C tiene m filas y n columnas.

¿Se puede mejorar este algoritmo?

Una segunda definición de **ContarCaminos**

Podemos reducir el número de llamadas recursivas:

ContarCaminos$(M[1 \dots n][1 \dots n],\ v_i,\ v_f,\ \ell)$

- **if** $\ell = 1$ **then return** $M[v_i, v_f]$
- **else**
  - $aux := 0$
  - **for** $v_j := 1$ **to** n **do**
    - **if** $M[v_i, v_j] = 1$ **then**
      - aux += **ContarCaminos**$(M,\ v_j,\ v_f,\ \ell - 1)$
  - **return** aux

¿Hay algún **problema** con este algoritmo?