Técnicas básicas de diseño de algoritmos

IIC2283

Técnicas de diseño de algoritmos

En este capítulo vamos a estudiar tres técnicas básicas para el diseño de algoritmos:

- Dividir para conquistar
- Programación dinámica
- Algoritmos codiciosos

Estas tres técnicas son útiles para desarrollar algoritmos en muchos escenarios.

Dividir para conquistar

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

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($S_1, \ldots, S_\ell$)

Dividir para conquistar

- En **DividirParaConquistar** k es un constante que indica cuando el tamaño de una entrada w es considerado pequeño: $|w| \leq k$
- Las entradas pequeñas son solucionas utilizado un algoritmo diseñado para ellas: **InstanciasPequeñas**
  - En general este algoritmo es sencillo y ejecuta un número pequeño de operaciones
- Si el tamaño de una entrada w no es pequeño ($|w| > k$), entonces w es dividido es una secuencia $w_1, \ldots, w_\ell$ de entradas de menor tamaño para **DividirParaConquistar**

Dividir para conquistar

- Para cada $i \in \{1, \ldots, \ell\}$ la llamada **DividirParaConquistar**(w_i) resuelve el problema para la entrada w_i
  - El resultado de esta llamada es almacenado en S_i
- Finalmente la llamada **Combinar**$(S_1, \ldots, S_\ell)$ combina los resultados $S_1, \ldots, S_\ell$ para obtener el resultados para w

Hemos visto varios ejemplos de esta técnica: algoritmo de multiplicación de Karatsuba, búsqueda binaria y Quicksort

- Vamos a ver como utilizar esta técnica para obtener un algoritmo eficiente para la multiplicación de matrices cuadradas

Multiplicación de matrices cuadradas

Sean $A_{n\times n}$ y $B_{n\times n}$ dos matrices de números enteros.

Queremos construir un algoritmo para calcular $A \cdot B$

- Medimos la complejidad del algoritmo en términos del número de sumas y multiplicaciones de números enteros, a las cuales asignamos costo 1

Ejercicio

Muestre que el algoritmo usual para multiplicar matrices cuadradas en el peor caso es $\Theta(n^3)$

Un cota inferior para la multiplicación de matrices

Sea $\mathcal{A}$ un algoritmo para calcular la multiplicación de matrices cuadradas de números enteros, y cuyas operaciones básicas son la suma y la multiplicación de números enteros.

Proposición

$\mathcal{A}$ debe realizar al menos n^2 sumas y multiplicaciones de números enteros para calcular la multiplicación de dos matrices de $n \times n$

Demostración: Utilizamos la técnica basada en la mejor estrategia del adversario

- En este caso el adversario demuestra que el algoritmo es incorrecto si realiza menos de n^2 operaciones básicas

Un cota inferior para la multiplicación de matrices

Para aplicar la técnica basada en la mejor estrategia del adversario primero debemos dar una caracterización del algoritmo $\mathcal{A}$

- De manera más general, una caracterización de los algoritmos que multiplican matrices cuadradas y tienen como operaciones básicas la suma y multiplicación de números enteros

Para calcular la multiplicación de dos matrices de $n \times n$ el algoritmo $\mathcal{A}$ debe calcular una secuencia $E_1, \ldots, E_k$ de expresiones aritméticas.

- El valor de k depende de n
- Cada una de las expresiones E_i incluye sumas y multiplicaciones de elementos de A o B o del resultado de otras expresiones
  - Por ejemplo, $E_1 = A[1,1] \cdot B[1,1]$ y $E_2 = E_1 + A[1,2] \cdot B[2,1]$

El uso de expresiones aritméticas es la clave para caracterizar $\mathcal{A}$

Un cota inferior para la multiplicación de matrices

Suponga que $\mathcal{A}$ puede calcular la multiplicación de dos matrices de $n \times n$ realizando $n^2 - 1$ operaciones básicas.

- ¿Por qué consideramos $n^2 - 1$ y no otra función $f(n)$ tal que $f(n) < n^2$?

El adversario elige dos matrices $A_{n \times n}$ y $B_{n \times n}$ tales que $A[i,j] = B[i,j] = 1$ para todo $i, j \in \{1, \ldots, n\}$, y le pide a $\mathcal{A}$ que calcule $A \cdot B$

$\mathcal{A}$ responde con una matriz $C_{n \times n}$ tal que $C[i,j] = n$ para todo $i, j \in \{1, \ldots, n\}$

- Suponemos que el algoritmo $\mathcal{A}$ es correcto

Un cota inferior para la multiplicación de matrices

Supongamos que para construir C el algoritmo $\mathcal{A}$ calcula las expresiones aritméticas $E_1, \ldots, E_k$

- Cada una de estas expresiones menciona al menos un símbolo $+$ o $\cdot$, por lo que $k \leq n^2 - 1$

Para cada $i \in \{1, \ldots, k\}$ suponga que hay $e_i \geq 1$ ocurrencias de símbolos $+$ y $\cdot$ en la expresión aritmética E_i

Ejercicio

Demuestre que para cada $i \in \{1, \ldots, k\}$ hay $e_i + 1$ ocurrencias de enteros en E_i

Un cota inferior para la multiplicación de matrices

Dado que $\mathcal{A}$ realiza $n^2 - 1$ operaciones para calcular C tenemos que:

$$\sum_{i=1}^{k} e_i \quad = \quad n^2 - 1$$

Tenemos entonces que:

$$\sum_{i=1}^{k} (e_i + 1) \;=\; n^2 - 1 + k < 2 \cdot n^2 - 1$$

Por el lema concluimos que existe al menos un elemento de A o B que no es utilizado en las expresiones aritméticas

Un cota inferior para la multiplicación de matrices

Sin perdida de generalidad suponemos que $A[r, s]$ no es utilizado en las expresiones aritméticas $E_1, \ldots, E_k$, donde $r, s \in \{1, \ldots, n\}$

El adversario entonces construye una matriz $D_{n \times n}$ tal que:

$$D[i, j] \quad = \quad \begin{cases} 1 & i \neq r \text{ o } j \neq s \\ 2 & i = r \text{ y } j = s \end{cases}$$

y le pide al algoritmo que calcule $D \cdot B$

Dado que la matriz D difiere de A sólo en el valor en la posición (r, s), el algoritmo $\mathcal{A}$ retorna C como el resultado de $D \cdot A$

- Obtenemos una contradicción puesto que $C \neq D \cdot A$ □

Un algoritmo de multiplicación más rápido

Tenemos n^2 como una cota inferior para el número operaciones básicas necesarias para obtener la multiplicación de dos matrices $A_{n\times n}$ y $B_{n\times n}$

- Y tenemos un algoritmo para este problema que en el peor caso es $\Theta(n^3)$

¡Hay mucho espacio para mejorar!

- ¿Es posible tener un algoritmo para multiplicar matrices cuadradas que realice $f(n)$ operaciones básicas, donde $f(n) \in o(n^3)$?

En las siguientes transparencias vamos a dar una respuesta positiva a esta pregunta: el algoritmo de Strassen

El algoritmo de multiplicación de Strassen

Sean $A_{n\times n}$ y $B_{n\times n}$ dos matrices de números enteros

- Suponemos inicialmente que n es una potencia de 2

Dividimos A y B de la siguiente forma:

$$A = \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \qquad B = \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix}$$

donde $A_{i,j}$ y $B_{i,j}$ ($i = 1, 2$ y $j = 1, 2$) son matrices de $\frac{n}{2} \times \frac{n}{2}$

El algoritmo de multiplicación de Strassen

Dividimos $C = A \cdot B$ de la misma forma:

$$C = \begin{pmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{pmatrix}$$

donde $C_{i,j}$ ($i = 1, 2$ y $j = 1, 2$) es una matriz de $\frac{n}{2} \times \frac{n}{2}$

Tenemos entonces que:

$$\begin{array}{lcllcl} C_{1,1} & = & A_{1,1} \cdot B_{1,1} + A_{1,2} \cdot B_{2,1} & C_{2,1} & = & A_{2,1} \cdot B_{1,1} + A_{2,2} \cdot B_{2,1} \\ C_{1,2} & = & A_{1,1} \cdot B_{1,2} + A_{1,2} \cdot B_{2,2} & C_{2,2} & = & A_{2,1} \cdot B_{1,2} + A_{2,2} \cdot B_{2,2} \end{array}$$

El cálculo directo de C requiere de 8 multiplicaciones de matrices de $\frac{n}{2} \times \frac{n}{2}$

- La idea clave del algoritmo de Strassen es bajar este número a 7

El algoritmo de multiplicación de Strassen

El algoritmo de Strassen calcula las siguientes multiplicaciones:

$$
\begin{aligned}
D_1 &= (A_{1,1} + A_{2,2}) \cdot (B_{1,1} + B_{2,2}) \\
D_2 &= (A_{2,1} + A_{2,2}) \cdot B_{1,1} \\
D_3 &= A_{1,1} \cdot (B_{1,2} - B_{2,2}) \\
D_4 &= A_{2,2} \cdot (B_{2,1} - B_{1,1}) \\
D_5 &= (A_{1,1} + A_{1,2}) \cdot B_{2,2} \\
D_6 &= (A_{2,1} - A_{1,1}) \cdot (B_{1,1} + B_{1,2}) \\
D_7 &= (A_{1,2} - A_{2,2}) \cdot (B_{2,1} + B_{2,2})
\end{aligned}
$$

El algoritmo de multiplicación de Strassen

Con estas multiplicaciones podemos calcular las matrices que forman C:

$$
\begin{aligned}
&D_1 + D_4 - D_5 + D_7 \\
&= (A_{1,1} + A_{2,2}) \cdot (B_{1,1} + B_{2,2}) + A_{2,2} \cdot (B_{2,1} - B_{1,1}) - \\
&\quad (A_{1,1} + A_{1,2}) \cdot B_{2,2} + (A_{1,2} - A_{2,2}) \cdot (B_{2,1} + B_{2,2}) \\
&= A_{1,1} \cdot B_{1,1} + A_{1,1} \cdot B_{2,2} + A_{2,2} \cdot B_{1,1} + A_{2,2} \cdot B_{2,2} + A_{2,2} \cdot B_{2,1} - A_{2,2} \cdot B_{1,1} - \\
&\quad A_{1,1} \cdot B_{2,2} - A_{1,2} \cdot B_{2,2} + A_{1,2} \cdot B_{2,1} + A_{1,2} \cdot B_{2,2} - A_{2,2} \cdot B_{2,1} - A_{2,2} \cdot B_{2,2} \\
&= A_{1,1} \cdot B_{1,1} + A_{1,2} \cdot B_{2,1} \\
&= C_{1,1}
\end{aligned}
$$

$$
\begin{aligned}
&D_3 + D_5 \\
&= A_{1,1} \cdot (B_{1,2} - B_{2,2}) + (A_{1,1} + A_{1,2}) \cdot B_{2,2} \\
&= A_{1,1} \cdot B_{1,2} - A_{1,1} \cdot B_{2,2} + A_{1,1} \cdot B_{2,2} + A_{1,2} \cdot B_{2,2} \\
&= A_{1,1} \cdot B_{1,2} + A_{1,2} \cdot B_{2,2} \\
&= C_{1,2}
\end{aligned}
$$

El algoritmo de multiplicación de Strassen

$$
\begin{aligned}
&D_2 + D_4 \\
&= (A_{2,1} + A_{2,2}) \cdot B_{1,1} + A_{2,2} \cdot (B_{2,1} - B_{1,1}) \\
&= A_{2,1} \cdot B_{1,1} + A_{2,2} \cdot B_{1,1} + A_{2,2} \cdot B_{2,1} - A_{2,2} \cdot B_{1,1} \\
&= A_{2,1} \cdot B_{1,1} + A_{2,2} \cdot B_{2,1} \\
&= C_{2,1}
\end{aligned}
$$

$$
\begin{aligned}
&D_1 - D_2 + D_3 + D_6 \\
&= (A_{1,1} + A_{2,2}) \cdot (B_{1,1} + B_{2,2}) - (A_{2,1} + A_{2,2}) \cdot B_{1,1} + \\
&\quad A_{1,1} \cdot (B_{1,2} - B_{2,2}) + (A_{2,1} - A_{1,1}) \cdot (B_{1,1} + B_{1,2}) \\
&= A_{1,1} \cdot B_{1,1} + A_{1,1} \cdot B_{2,2} + A_{2,2} \cdot B_{1,1} + A_{2,2} \cdot B_{2,2} - A_{2,1} \cdot B_{1,1} - A_{2,2} \cdot B_{1,1} + \\
&\quad A_{1,1} \cdot B_{1,2} - A_{1,1} \cdot B_{2,2} + A_{2,1} \cdot B_{1,1} + A_{2,1} \cdot B_{1,2} - A_{1,1} \cdot B_{1,1} - A_{1,1} \cdot B_{1,2} \\
&= A_{2,1} \cdot B_{1,2} + A_{2,2} \cdot B_{2,2} \\
&= C_{2,2}
\end{aligned}
$$

La complejidad del algoritmo de Strassen

¿Cuál es el peor caso para el algoritmo de Strassen?

- El algoritmo realiza la misma cantidad de sumas y multiplicaciones de números enteros para todas las matrices de entrada de $n \times n$

Sea $T(n)$ es número de sumas y multiplicaciones de números enteros realizadas por el algoritmo de Strassen con matrices de entradas de $n \times n$

Tenemos que:

$$T(n) = \begin{cases} 1 & n = 1 \\ 7 \cdot T(\frac{n}{2}) + c \cdot n^2 & n > 1 \end{cases}$$

donde $c \in \mathbb{R}^+$. ¿Por qué?

La complejidad del algoritmo de Strassen

La ecuación de recurrencia anterior sólo fue definida para n potencia de 2, pero podemos extenderla para considerar cualquier valor de n:

$$T(n) = \begin{cases} 1 & n = 1 \\ 7 \cdot T(\lfloor \frac{n}{2} \rfloor) + c \cdot n^2 & n > 1 \end{cases}$$

Dado que $\log_2(7) > 2.8$, tenemos que $c \cdot n^2 \in O(n^{\log_2(7)-0.5})$, por lo que usando el Teorema Maestro obtenemos que $T(n) \in \Theta(n^{\log_2(7)})$

En particular concluimos que existe $d \in \mathbb{R}^+$ y $n_0 \in \mathbb{N}$ tales que para todo $n \geq n_0$ que sea potencia de 2 se tiene que $T(n) \leq d \cdot n^{\log_2(7)}$

La complejidad del algoritmo de Strassen

Tenemos entonces un algoritmo para multiplicar matrices cuadradas que funciona más rápido que el algoritmo usual.

¿Pero como usamos este algoritmo cuando n no es potencia de 2?

- Vamos a responder esta pregunta en las siguientes transparencias

Extendiendo el algoritmo de Strassen

Sean $A_{n\times n}$ y $B_{n\times n}$ dos matrices de números enteros donde n puede no ser una potencia de 2

- Tenemos que $2^k \leq n < 2^{k+1}$ para $k \geq 1$

Si $2^k = n$ calculamos $A \cdot B$ con el algoritmo de Strassen. Si $2^k < n$, entonces a partir de A y B definimos matrices $D_{2^{k+1}\times 2^{k+1}}$ y $E_{2^{k+1}\times 2^{k+1}}$ como sigue:

$$D[i,j] = \begin{cases} A[i,j] & 1 \leq i \leq n \text{ y } 1 \leq j \leq n \\ 0 & n+1 \leq i \leq 2^k \text{ o } n+1 \leq j \leq 2^{k+1} \end{cases}$$

$$E[i,j] = \begin{cases} B[i,j] & 1 \leq i \leq n \text{ y } 1 \leq j \leq n \\ 0 & n+1 \leq i \leq 2^k \text{ o } n+1 \leq j \leq 2^{k+1} \end{cases}$$

Extendiendo el algoritmo de Strassen

Utilizando el algoritmo de Strassen calculamos $F = D \cdot E$, y a partir de F definimos $C_{n \times n}$ como $C[i,j] = F[i,j]$ para $i, j \in \{1, \ldots, n\}$

Puesto que $C = A \cdot B$ tenemos un algoritmo general para calcular la multiplicación de matrices cuadradas.

Ejercicio

Demuestre que el algoritmo extendido de Strassen en el peor caso es $O(n^{\log_2(7)})$ considerando como operaciones básicas la suma y multiplicación de enteros.

- ¿Hay un peor caso para este algoritmo?
- ¿Qué sucede si incluye en el tiempo de ejecución la construcción de las matrices C y D? ¿Cambia el orden del algoritmo?