Algoritmos divide y vencerás

José de Jesús Lavalle Martínez

Benemérita Universidad Autónoma de Puebla
Facultad de Ciencias de la Computación
Maestría en Ciencias de la Computación
Análisis y Diseño de Algoritmos
MCOM 20300

Contenido

Introducción

- Divide y vencerás es una técnica de diseño de algoritmos que consiste en descomponer la instancia a resolver en varias subinstancias más pequeñas del mismo problema.

Introducción

- Resolviendo sucesiva e independientemente cada una de estas subinstancias, y luego combinar las subsoluciones así obtenidas para obtener la solución de la instancia original.

Introducción

- Dos preguntas que naturalmente vienen a la mente son:

Introducción

- Dos preguntas que naturalmente vienen a la mente son:
  1. “¿Por qué alguien querría hacer esto?” y

Introducción

- Dos preguntas que naturalmente vienen a la mente son:
  1. "¿Por qué alguien querría hacer esto?" y
  2. "¿Cómo deberíamos resolver las subinstancias?"

Introducción

2. "¿Cómo deberíamos resolver las subinstancias?"

- La eficacia de la técnica divide y vencerás radica en la respuesta a esta última pregunta.

Multiplicando enteros grandes I

- Considere el problema de multiplicar números enteros grandes.

Multiplicando enteros grandes I

- El algoritmo clásico que la mayoría de nosotros aprendemos en la escuela requiere un tiempo en $\Theta(n^2)$ para multiplicar dos números de n cifras.

Multiplicando enteros grandes I

- Estamos tan acostumbrados a este algoritmo que quizás ni siquiera cuestionaste su optimalidad.

Multiplicando enteros grandes I

- ¿Podemos hacerlo mejor?

Multiplicando enteros grandes I

- La multiplicación *à la russe* no ofrece ninguna mejora en el tiempo de ejecución.

Multiplicando enteros grandes II

- Otro algoritmo que usa la técnica "divide y vencerás" consiste en reducir la multiplicación de dos números de n cifras a cuatro multiplicaciones de números de $\frac{n}{2}$ cifras.

Multiplicando enteros grandes II

- Desafortunadamente, el algoritmo resultante tampoco produce ninguna mejora sobre el algoritmo de multiplicación clásico a menos que seamos más inteligentes.

Multiplicando enteros grandes II

- Para superar el algoritmo clásico y, por tanto, apreciar plenamente las virtudes de divide y vencerás, debemos encontrar una manera de reducir la multiplicación original no a cuatro, sino a *tres* multiplicaciones de la mitad del tamaño.

Multiplicando enteros grandes III

- Ilustramos el proceso con el ejemplo usado en la Sección 1.2 del libro de texto: la multiplicación de 981 por 1234.

Multiplicando enteros grandes III

- Primero rellenamos el operando más corto con un cero no significativo para que tenga la misma longitud que el más largo; así 981 se convierte en 0981.

Multiplicando enteros grandes III

- Luego dividimos cada operando en dos mitades: 0981 da lugar a $w = 09$ y $x = 81$, y 1234 a $y = 12$ y $z = 34$.

Multiplicando enteros grandes III

- Luego dividimos cada operando en dos mitades: 0981 da lugar a $w = 09$ y $x = 81$, y 1234 a $y = 12$ y $z = 34$.
- Observe que $981 = 10^2 w + x$ y $1234 = 10^2 y + z$.

Multiplicando enteros grandes III

- Por lo tanto, el producto requerido se puede calcular como

$$\begin{aligned} 981 \times 1234 &= (10^2 w + x) \times (10^2 y + z) \\ &= 10^4 wy + 10^2(wz + xy) + xz \\ &= 1080000 + 127800 + 2754 = 1021554. \end{aligned}$$

Multiplicando enteros grandes IV

- Si cree que simplemente hemos reformulado el algoritmo de la Sección 1.2 del libro de texto con más símbolos, está perfectamente en lo cierto.

Multiplicando enteros grandes IV

- El procedimiento anterior todavía necesita cuatro multiplicaciones de la mitad del tamaño: wy, wz, xy y xz.

Multiplicando enteros grandes IV

- La observación clave es que no es necesario calcular wz ni xy; todo lo que realmente necesitamos es la *suma* de estos dos términos.

Multiplicando enteros grandes IV

- ¿Es posible obtener $wz + xy$ al costo de una sola multiplicación?

Multiplicando enteros grandes IV

- Esto parece imposible hasta que recordemos que también necesitamos los valores de wy y xz para aplicar la fórmula anterior.

Multiplicando enteros grandes IV

- Esto parece imposible hasta que recordemos que también necesitamos los valores de wy y xz para aplicar la fórmula anterior.
- Con esto en mente, considere el producto

$$r = (w + x) \times (y + z) = wy + (wz + xy) + xz.$$

Multiplicando enteros grandes V

- Después de una sola multiplicación, obtenemos la suma de los tres términos necesarios para calcular el producto deseado.
- Esto sugiere proceder de la siguiente manera.

$$
\begin{aligned}
p &= wy = 09 \times 12 = 108 \\
q &= xz = 81 \times 34 = 2754 \\
r &= (w + x) \times (y + z) = 90 \times 46 = 4140
\end{aligned}
$$

- y finalmente

$$
\begin{aligned}
981 \times 1234 &= 10^4 p + 10^2 (r - p - q) + q \\
&= 1080000 + 127800 + 2754 = 1210554.
\end{aligned}
$$

Multiplicando enteros grandes VI

- Así, el producto de 981 y 1234 se puede reducir a tres multiplicaciones de números de dos cifras ($09 \times 12, 81 \times 34$ y 90×46) junto con un cierto número de corrimientos (multiplicaciones por potencias de 10), adiciones y restas.

Multiplicando enteros grandes VI

- Sin duda, el número de sumas -contar las restas como si fueran sumas- es mayor que con el algoritmo original divide y vencerás de la Sección 1.2 del libro de texto.

Multiplicando enteros grandes VI

- ¿Vale la pena realizar cuatro sumas más para ahorrar una multiplicación?

Multiplicando enteros grandes VI

- ¿Vale la pena realizar cuatro sumas más para ahorrar una multiplicación?
- La respuesta es no cuando multiplicamos números pequeños como los de nuestro ejemplo.

Multiplicando enteros grandes VI

- Sin embargo, vale la pena cuando los números que se van a multiplicar son grandes, y lo es cada vez más cuando los números aumentan.

Multiplicando enteros grandes VII

- Cuando los operandos son grandes, el tiempo requerido para las adiciones y corrimientos se vuelve insignificante en comparación con el tiempo que toma una sola multiplicación.

Multiplicando enteros grandes VII

- Por tanto, parece razonable esperar que reducir cuatro multiplicaciones a tres nos permitirá reducir el 25% del tiempo de cálculo necesario para las multiplicaciones grandes.

Multiplicando enteros grandes VII

- Como veremos, nuestro ahorro será significativamente mejor.

Multiplicando enteros grandes VII

- Para ayudar a comprender lo que hemos logrado, suponga que una implementación dada del algoritmo de multiplicación clásico requiere un tiempo $h(n) = cn^2$ para multiplicar dos números de n cifras, por alguna constante c que depende de la implementación (esta es una simplificación ya que en realidad el tiempo requerido tendría una forma más complicada, como $cn^2 + bn + a$).

Multiplicando enteros grandes VIII

- De manera similar, sea $g(n)$ el tiempo que tarda el algoritmo divide y venceras para multiplicar dos números de n cifras, sin contar el tiempo necesario para realizar las tres multiplicaciones de la mitad del tamaño.

Multiplicando enteros grandes VIII

- En otras palabras, $g(n)$ es el tiempo necesario para las adiciones, corrimientos y varias operaciones generales.

Multiplicando enteros grandes VIII

- Es fácil implementar estas operaciones de modo que $g(n) \in \Theta(n)$.

Multiplicando enteros grandes VIII

- Ignore por el momento lo que sucede si n es impar o si los números no tienen la misma longitud.

Multiplicando enteros grandes IX

- Si cada una de las tres multiplicaciones de la mitad del tamaño se realiza mediante el algoritmo clásico, el tiempo necesario para multiplicar dos números de n cifras es

$$3h(n/2) + g(n) = 3c(n/2)^2 + g(n) = \frac{3}{4}cn^2 + g(n) = \frac{3}{4}h(n) + g(n).$$

Multiplicando enteros grandes IX

- Como $h(n) \in \Theta(n^2)$ y $g(n) \in \Theta(n)$, el término $g(n)$ es insignificante en comparación con $\frac{3}{4}h(n)$ cuando n es lo suficientemente grande, lo que significa que hemos ganado aproximadamente un 25% en velocidad en comparación con el algoritmo clásico, como anticipamos.

Multiplicando enteros grandes IX

- Aunque esta mejora no debe ser despreciada, no hemos logrado cambiar el orden del tiempo requerido: el nuevo algoritmo aún toma tiempo cuadrático.

Multiplicando enteros grandes X

- Para hacerlo mejor, volvemos a la pregunta planteada en el párrafo inicial: ¿cómo se deben resolver las subinstancias?

Multiplicando enteros grandes X

- Si son pequeños, el algoritmo clásico puede ser la mejor manera de proceder.

Multiplicando enteros grandes X

- Sin embargo, cuando las subinstancias son lo suficientemente grandes, ¿no sería mejor utilizar nuestro nuevo algoritmo de forma recursiva?

Multiplicando enteros grandes X

- ¡La idea es análoga a beneficiarse de una cuenta bancaria que capitaliza los pagos de intereses!

Multiplicando enteros grandes X

- Cuando hacemos esto, obtenemos un algoritmo que puede multiplicar dos números de n cifras en un tiempo $t(n) = 3t(n/2) + g(n)$ cuando n es par y suficientemente grande.

Multiplicando enteros grandes XI

- Esta recurrencia es similar a la que estudiamos en el Ejemplo 10 de las notas del curso, resolviéndola obtuvimos que $t(n) \in \Theta(n^{\log 3} | n$ es potencia de 2).

Multiplicando enteros grandes XI

- Tenemos que contentarnos con la notación asintótica condicional porque todavía no hemos abordado la cuestión de cómo multiplicar números de longitud impar; vea el problema 7.1 del libro de texto.

Multiplicando enteros grandes XI

- Dado que $\lg 3 \approx 1.585$ es menor que 2, este algoritmo puede multiplicar dos enteros grandes mucho más rápido que el algoritmo de multiplicación clásico, y cuanto mayor sea n, más vale la pena tener esta mejora.

Multiplicando enteros grandes XI

- Una buena implementación probablemente no utilizará la base 10, sino la base más grande para la que el hardware permita que se multipliquen directamente dos “dígitos”.

Multiplicando enteros grandes XII

- Un factor importante en la eficacia práctica de este enfoque de la multiplicación, y de hecho de cualquier algoritmo divide y vencerás, es saber cuándo dejar de dividir las instancias y usar el algoritmo clásico en su lugar.

Multiplicando enteros grandes XII

- Aunque el enfoque divide y vencerás se vuelve más valioso a medida que la instancia a resolver se hace más grande, de hecho puede ser más lento que el algoritmo clásico en instancias que son demasiado pequeñas.

Multiplicando enteros grandes XII

- Por lo tanto, un algoritmo divide y vencerás debe evitar proceder de manera recursiva cuando el tamaño de las subinstancias ya no lo justifique.

Multiplicando enteros grandes XIII

- En aras de la simplicidad, hasta ahora se han ocultado varios temas importantes.

Multiplicando enteros grandes XIII

- ¿Cómo nos ocupamos de los números de longitud impar?

Multiplicando enteros grandes XIII

- Aunque ambas mitades del multiplicador y el multiplicando sean de tamaño $n/2$, puede suceder que su suma se desborde y sea de tamaño 1 mayor.

Multiplicando enteros grandes XIII

- Por lo tanto, fue un poco incorrecto afirmar que $r = (w + x) \times (y + z)$ implica una multiplicación de la mitad del tamaño.

Multiplicando enteros grandes XIII

- ¿Cómo afecta esto al análisis del tiempo de ejecución? ¿Cómo multiplicamos dos números de diferentes tamaños?

Multiplicando enteros grandes XIII

- ¿Hay operaciones aritméticas distintas de la multiplicación que podamos manejar de manera más eficiente que usando algoritmos clásicos?

Multiplicando enteros grandes XIV

- Los números de longitud impar se multiplican fácilmente dividiéndolos tan cerca de la mitad como sea posible: un número de n cifras se divide en un número de $\lfloor n/2 \rfloor$ cifras y un número de $\lceil n/2 \rceil$ cifras.

Multiplicando enteros grandes XIV

- La segunda pregunta es más complicada.

Multiplicando enteros grandes XIV

- Considere la posibilidad de multiplicar 5678 por 6789. Nuestro algoritmo divide los operandos en $w = 56, x = 78, y = 67$ y $z = 89$.

Multiplicando enteros grandes XIV

- Considere la posibilidad de multiplicar 5678 por 6789. Nuestro algoritmo divide los operandos en $w = 56, x = 78, y = 67$ y $z = 89$.
- Las tres multiplicaciones, de la mitad del tamaño, involucradas son

$$p = wy = 56 \times 67$$
$$q = xz = 78 \times 89, \text{ y}$$
$$r = (w + x) \times (y + z) = 134 \times 156.$$

Multiplicando enteros grandes XV

- La tercera multiplicación implica números de tres cifras y, por lo tanto, no es realmente la mitad del tamaño en comparación con la multiplicación original de números de cuatro cifras.

Multiplicando enteros grandes XV

- Sin embargo, el tamaño de $w + x$ y de $y + z$ no puede exceder $1 + \lceil n/2 \rceil$.

Multiplicando enteros grandes XV

- Para simplificar el análisis, sea $t(n)$ el tiempo que tarda este algoritmo en el peor de los casos para multiplicar dos números de tamaño como máximo n (en lugar de exactamente n).

Multiplicando enteros grandes XV

- Por definición, $t(n)$ es una función no decreciente.

Multiplicando enteros grandes XVI

- Cuando n es lo suficientemente grande, nuestro algoritmo reduce la multiplicación de dos números de tamaño como máximo n a tres multiplicaciones más pequeñas $p = wy, q = xz$ y $r = (w + x) \times (y + z)$ de tamaños como máximo $\lfloor n/2 \rfloor, \lceil n/2 \rceil$ y $1 + \lceil n/2 \rceil$, respectivamente, además de manipulaciones fáciles que toman un tiempo en $O(n)$.

Multiplicando enteros grandes XVI

- Por tanto, existe una constante positiva c tal que

$$t(n) \leq t(\lfloor n/2 \rfloor) + t(\lceil n/2 \rceil) + t(1 + \lceil n/2 \rceil) + cn$$

para todo n suficientemente grande.

Multiplicando enteros grandes XVI

- Ésta es precisamente la recurrencia que estudiamos en el ejemplo 4.7.14 del libro de texto, que produce el ahora familiar $t(n) \in O(n^{\lg 3})$.

Multiplicando enteros grandes XVI

- Por tanto, siempre es posible multiplicar números de n cifras en un tiempo en $O(n^{\lg 3})$.

Multiplicando enteros grandes XVII

- Un análisis del peor caso de este algoritmo muestra que, de hecho, $t(n) \in \Theta(n^{\lg 3})$, pero esto es de interés limitado porque existen algoritmos de multiplicación aún más rápidos; consulte los problemas 7.2 y 7.3 del libro de texto.

Multiplicando enteros grandes XVII

- Regresando a la cuestión de multiplicar números de diferente tamaño, sean u y v números enteros de tamaño m y n, respectivamente.

Multiplicando enteros grandes XVII

- Si m y n están dentro de un factor de dos entre sí, es mejor rellenar el operando más pequeño con ceros no significativos para que tenga la misma longitud que el otro operando, como hicimos cuando multiplicamos 981 por 1234.

Multiplicando enteros grandes XVIII

- Sin embargo, este enfoque debe desalentarse cuando un operando es mucho más grande que el otro.

Multiplicando enteros grandes XVIII

- ¡Incluso podría ser peor que usar el algoritmo de multiplicación clásico!

Multiplicando enteros grandes XVIII

- Sin pérdida de generalidad, suponga que $m \leq n$. El algoritmo divide y vencerás que se usa con el relleno y el algoritmo clásico toman tiempo en $\Theta(n^{\lg 3})$ y $\Theta(mn)$, respectivamente, para calcular el producto de u y v.

Multiplicando enteros grandes XVIII

- Teniendo en cuenta que es probable que la constante oculta del primero sea mayor que la del segundo, vemos que divide y vencerás con relleno es más lento que el algoritmo clásico cuando $m < n^{\lg(3/2)}$, y por lo tanto en particular cuando $m < \sqrt{n}$.

Multiplicando enteros grandes XIX

- Sin embargo, es sencillo combinar ambos algoritmos para obtener un algoritmo realmente mejor.

Multiplicando enteros grandes XIX

- La idea es dividir el operando v más grande en bloques de tamaño m y usar el algoritmo divide y vencerás para multiplicar u por cada bloque de v, de modo que el algoritmo divide y vencerás se use para multiplicar pares de operandos del mismo tamaño.

Multiplicando enteros grandes XIX

- El producto final de u y v se obtiene entonces fácilmente mediante simples adiciones y corrimientos.

Multiplicando enteros grandes XIX

- El tiempo de ejecución total está dominado por la necesidad de realizar $\lceil n/m \rceil$ multiplicaciones de números de m cifras.

Multiplicando enteros grandes XX

- Dado que cada una de estas multiplicaciones más pequeñas toma un tiempo en $\Theta(m^{\lg 3})$ y dado que $\lceil n/m \rceil \in \Theta(n/m)$, el tiempo total de ejecución para multiplicar un número de n cifras por un número de m cifras está en $\Theta(nm^{\lg(3/2)})$ cuando $m \leq n$.

Multiplicando enteros grandes XX

- La multiplicación no es la única operación interesante que involucra números enteros grandes. La exponenciación modular es crucial para la criptografía moderna; consulte la Sección 7.8 del libro de texto.

Multiplicando enteros grandes XXI

- La división de enteros, las operaciones de módulo y el cálculo de la parte entera de una raíz cuadrada se pueden realizar en un tiempo cuyo orden es el mismo que el requerido para la multiplicación; consulte la Sección 12.4 del libro de texto.

Multiplicando enteros grandes XXI

- Algunas otras operaciones importantes, como calcular el máximo común divisor, pueden ser intrínsecamente más difíciles de calcular; no se tratan aquí.

La plantilla general I

- La multiplicación de números enteros grandes no es un ejemplo aislado del beneficio que se obtiene del enfoque de divide y vencerás.

La plantilla general I

- Considere un problema arbitrario y suponga que ADHOC es un algoritmo simple capaz de resolver el problema.

La plantilla general I

- Pedimos a ADHOC que sea eficiente en instancias pequeñas, pero su rendimiento en instancias grandes no es motivo de preocupación.

La plantilla general I

- Lo llamamos el subalgoritmo básico.

La plantilla general I

- El algoritmo de multiplicación clásico es un ejemplo de un subalgoritmo básico.

La plantilla general II

La plantilla general para los algoritmos divide y vencerás es la siguiente.

function $\mathrm{DC}(x)$

1 **if** x es suficientemente pequeño o simple
2 **then** $\textsc{adhoc}(x)$
3 descomponga x en instancias más pequeñas $x_1, x_2, \ldots, x_l$
4 **for** $i \leftarrow 1$ **to** l
5 **do** $y_i \leftarrow \mathrm{DC}(x_i)$
6 recombine las y_i para obtener una solución y para x
7 **return** y

La plantilla general III

- Algunos algoritmos divide y vencerás no siguen este esquema exactamente: por ejemplo, podrían requerir que la primera subinstancia se resuelva antes de formular la segunda subinstancia; consulte la Sección 7.5 del libro de texto.

La plantilla general III

- El número de subinstancias, l, suele ser pequeño e independiente de la instancia particular a resolver.

La plantilla general III

- Cuando $l = 1$, no tiene mucho sentido "descomponer x en una instancia más pequeña x_1" y es difícil justificar llamar a la técnica divide y vencerás.

La plantilla general IV

- Sin embargo, tiene sentido reducir la solución de una instancia grande a la de una más pequeña.

La plantilla general IV

- Divide y vencerás se conoce con el nombre de *simplificación* en este caso; consulte las Secciones 7.3 y 7.7 del libro de texto.

La plantilla general IV

- Cuando se usa la simplificación, a veces es posible reemplazar la recursividad inherente a divide y vencerás por un ciclo iterativo.

La plantilla general IV

- Implementado en un lenguaje convencional como Pascal en una máquina convencional que ejecuta un compilador poco sofisticado, es probable que un algoritmo iterativo sea algo más rápido que la versión recursiva, aunque solo por un factor multiplicativo constante.

La plantilla general V

- Por otro lado, es posible ahorrar una cantidad sustancial de almacenamiento de esta manera: para una instancia de tamaño n, el algoritmo recursivo usa una pila cuya profundidad a menudo está en $\Omega(\lg n)$ y en casos malos incluso en $\Omega(n)$.

La plantilla general V

- Para que valga la pena divide y vencerás se deben tomar en cuenta tres cosas.

La plantilla general V

- Para que valga la pena divide y vencerás se deben tomar en cuenta tres cosas.
  1. La decisión de cuándo utilizar el subalgoritmo básico en lugar de hacer llamadas recursivas debe tomarse con prudencia.

La plantilla general V

- Para que valga la pena divide y vencerás se deben tomar en cuenta tres cosas.

2. Debe ser posible descomponer una instancia en subinstancias y recombinar las subsoluciones de manera bastante eficiente.

La plantilla general V

- Para que valga la pena divide y vencerás se deben tomar en cuenta tres cosas.

3. Las subinstancias deben ser, en la medida de lo posible, aproximadamente del mismo tamaño.

La plantilla general VI

- La mayoría de los algoritmos divide y vencerás son tales que el tamaño de las l subinstancias es aproximadamente n/b para alguna constante b, donde n es el tamaño de la instancia original.

La plantilla general VI

- Por ejemplo, nuestro algoritmo divide y vencerás para multiplicar números enteros grandes necesita un tiempo en $\Theta(n)$ para descomponer la instancia original en tres subinstancias de aproximadamente la mitad del tamaño y recombinar las subsoluciones: $l = 3$ y $b = 2$.

La plantilla general VI

- El análisis del tiempo de ejecución de tales algoritmos de divide y vencerás es casi automático, gracias a los ejemplos 13 y 4.7.16 del libro de texto.

La plantilla general VII

- Sea $g(n)$ el tiempo requerido por DC en instancias de tamaño n, sin contar el tiempo necesario para las llamadas recursivas.

La plantilla general VII

- El tiempo total $t(n)$ que toma este algoritmo de divide y vencerás es algo así como

$$t(n) = lt(n \div b) + g(n)$$

siempre que n sea lo suficientemente grande.

La plantilla general VII

- Si existe un entero k tal que $g(n) \in \Theta(n^k)$, entonces se aplica el ejemplo 4.7.16 del libro de texto para concluir que

$$T(n) \in \begin{cases} \Theta(n^k) & \text{si } l < b^k \\ \Theta(n^k \log n) & \text{si } l = b^k \\ \Theta(n^{\log_b l}) & \text{si } l > b^k \end{cases} \quad (1)$$

La plantilla general VIII

- Las técnicas utilizadas en la Sección 4.7.6 y el Ejemplo 4.7.14 del libro de texto generalmente se aplican para producir la misma conclusión, incluso si algunas de las subinstancias son de un tamaño que difiere de $\lfloor n/b \rfloor$ como máximo en una constante aditiva, y en particular si alguna de las las subinstancias son de tamaño $\lceil n/b \rceil$.

La plantilla general VIII

- Como ejemplo, nuestro algoritmo divide y vencerás para la multiplicación de enteros grandes se caracteriza por $l = 3, b = 2$ y $k = 1$.

La plantilla general VIII

- Como $l > b^k$, se aplica el tercer caso y obtenemos inmediatamente que el algoritmo toma un tiempo en $\Theta(n^{\lg 3})$ sin necesidad de preocuparse por el hecho de que dos de las subinstancias son de tamaño $\lceil n/2 \rceil$ y $1 + \lceil n/2 \rceil$ en lugar de $\lfloor n/2 \rfloor$.

La plantilla general IX

- En casos más complicados cuando $g(n)$ no está en el orden exacto de un polinomio, puede aplicarse el problema 4.44 del libro de texto.

La plantilla general IX

- Queda por ver cómo determinar si dividir la instancia y hacer llamadas recursivas, o si la instancia es tan simple que es mejor invocar el subalgoritmo básico directamente.

La plantilla general IX

- Aunque esta elección no afecta el orden del tiempo de ejecución del algoritmo, también nos preocupa que la constante multiplicativa oculta en la notación Θ sea lo más pequeña posible.

La plantilla general IX

- Con la mayoría de los algoritmos divide y vencerás, esta decisión se basa en un umbral simple, generalmente denotado n_0.

La plantilla general IX

- El subalgoritmo básico se utiliza para resolver cualquier instancia cuyo tamaño no supere a n_0.

La plantilla general X

- Volvamos al problema de multiplicar números enteros grandes para ver por qué es importante la elección del umbral y cómo elegirlo.

La plantilla general X

- Para evitar nublar los aspectos esenciales, usamos una fórmula de recurrencia simplificada para el tiempo de ejecución del algoritmo divide y vencerás para multiplicar números enteros grandes:

$$T(n) = \begin{cases} h(n) & \text{si } n \leq n_0 \\ 3T(\lceil n/2 \rceil) + g(n) & \text{en otro caso,} \end{cases}$$

donde $h(n) \in \Theta(n^2)$ y $g(n) \in \Theta(n)$.

La plantilla general XI

- En aras de la argumentación, considere una implementación donde $h(n) = n^2$ microsegundos y $g(n) = 16n$ microsegundos.

La plantilla general XI

- Suponga que nos dan dos números de 5000 cifras para multiplicar.

La plantilla general XI

- Si el algoritmo divide y vencerás procede de forma recursiva hasta obtener subinstancias de tamaño 1, es decir, si $n_0 = 1$, se necesitan más de 41 segundos para calcular el producto.

La plantilla general XI

- Esto es ridículo, ya que los mismos números se pueden multiplicar en 25 segundos usando el algoritmo clásico.

La plantilla general XI

- El algoritmo clásico supera ligeramente al algoritmo divide y vencerás incluso para multiplicar números con hasta 32 789 cifras, cuando ambos algoritmos requieren más de un cuarto de hora de tiempo de cálculo ¡para una sola multiplicación!

La plantilla general XII

- ¿Debemos concluir que divide y vencerás nos permite pasar de un algoritmo cuadrático a un algoritmo cuyo tiempo de ejecución está en $\Theta(n^{\lg 3})$, pero solo a costa de un aumento de la constante oculta tan enorme que el nuevo algoritmo nunca es económico en instancias de tamaño razonable?

La plantilla general XII

- Afortunadamente no: para continuar con nuestro ejemplo, los números de 5000 cifras se pueden multiplicar en poco más de 6 segundos, siempre que elijamos el umbral n_0 inteligentemente; en este caso, $n_0 = 64$ es una buena opción.

La plantilla general XII

- Con el mismo umbral, se necesitan poco más de dos minutos para multiplicar dos números de 32 789 cifras.

La plantilla general XIII

- La elección del mejor umbral es complicada por el hecho de que el valor óptimo generalmente depende no solo del algoritmo en cuestión, sino también de la implementación particular.

La plantilla general XIII

- Además, en general, no existe un valor óptimo uniforme del umbral.

La plantilla general XIII

- En nuestro ejemplo, es mejor utilizar el algoritmo clásico para multiplicar números de 67 cifras, mientras que es mejor repetir una vez para multiplicar números de 66 cifras.

La plantilla general XIII

- Así, 67 es mejor que 64 como umbral en el primer caso, mientras que en el segundo caso ocurre lo contrario. En el futuro abusaremos del término “umbral óptimo” para significar casi óptimo.

La plantilla general XIV

- Entonces, ¿cómo elegiremos el umbral?

La plantilla general XIV

- Dada una implementación particular, el umbral óptimo se puede determinar empíricamente.

La plantilla general XIV

- Variamos el valor del umbral y el tamaño de las instancias utilizadas para nuestras pruebas y cronometramos la implementación en varios casos.

La plantilla general XIV

- A menudo es posible estimar un umbral óptimo tabulando los resultados de estas pruebas o dibujando algunos diagramas.

La plantilla general XIV

- Sin embargo, los cambios en el valor del umbral en un cierto rango pueden no tener ningún efecto sobre la eficiencia del algoritmo cuando sólo se consideran instancias de algún tamaño específico.

La plantilla general XV

- Por ejemplo, se necesita exactamente el mismo tiempo para multiplicar dos números de 5000 cifras cuando el umbral se establece entre 40 y 78, ya que cualquier valor de ese tipo para el umbral hace que la recursividad se detenga cuando las subinstancias alcanzan el tamaño 40, por debajo del tamaño 79, en el séptimo nivel de recursividad.

La plantilla general XV

- Sin embargo, estos umbrales no son equivalentes en general, ya que los números de 41 cifras tardan un 17% más en multiplicarse con el umbral establecido en 40 en lugar de en 64.

La plantilla general XV

- Por lo tanto, generalmente no es suficiente simplemente variar el umbral para una instancia cuyo tamaño permanece fijo.

La plantilla general XVI

- Este enfoque empírico puede requerir una cantidad considerable de tiempo informático (¡y humano!).

La plantilla general XVI

- Una vez les pedimos a los estudiantes de un curso de algoritmos que implementaran el algoritmo divide y vencerás para multiplicar números enteros grandes y que lo comparen con el algoritmo clásico.

La plantilla general XVI

- Varios grupos trataron de estimar empíricamente el umbral óptimo, ¡cada grupo utilizó en el intento más de 5000 dólares en tiempo de máquina!

La plantilla general XVI

- Por otro lado, rara vez es posible un cálculo puramente teórico del umbral óptimo, dado que varía de una implementación a otra.

La plantilla general XVI

- El enfoque híbrido, que recomendamos, consiste en determinar teóricamente la forma de las ecuaciones de recurrencia y luego encontrar empíricamente los valores de las constantes utilizadas en estas ecuaciones para la implementación en cuestión.

La plantilla general XVII

- El umbral óptimo se puede estimar entonces encontrando el tamaño n de la instancia para la que no importa si aplicamos el algoritmo clásico directamente o si continuamos con un nivel más de recursividad; vea el problema 7.8 del libro de texto.

La plantilla general XVII

- Es por eso que elegimos $n_0 = 64$: el algoritmo de multiplicación clásico requiere $h(64) = 64^2 = 4096$ microsegundos para multiplicar dos números de 64 cifras.

La plantilla general XVII

- Mientras que si usamos un nivel más de recursividad en el enfoque divide y vencerás, la misma multiplicación requiere $g(64) = 16 \times 64 = 1024$ microsegundos además de tres multiplicaciones de números de 32 cifras por el algoritmo clásico, a un costo de $h(32) = 32^2 = 1024$ microsegundos cada uno, para el mismo total de $3h(32) + g(64) = 4096$ microsegundos.

La plantilla general XVIII

- Surge una dificultad práctica con esta técnica híbrida.

La plantilla general XVIII

- Aunque el algoritmo de multiplicación clásico requiere tiempo cuadrático, fue una simplificación excesiva afirmar que $h(n) = cn^2$ para alguna constante c que depende de la implementación.

La plantilla general XVIII

- Es más probable que existan tres constantes a, b y c tales que $h(n) = cn^2 + bn + a$.

La plantilla general XVIII

- Aunque $bn + a$ se vuelve insignificante en comparación con cn^2 cuando n es grande, el algoritmo clásico se usa de hecho precisamente en instancias de tamaño moderado.

La plantilla general XIX

- Por lo tanto, generalmente es insuficiente estimar simplemente la constante de orden superior c.

La plantilla general XIX

- En cambio, es necesario medir $h(n)$ varias veces para varios valores diferentes de n para estimar todas las constantes necesarias.

La plantilla general XIX

- La misma observación se aplica a $g(n)$.

Ejercicios

1. Equipo 1: Binarysearch, sección 7.3 del libro de texto.
2. Equipo 2: Mergesort, sección 7.4.1 del libro de texto.
3. Equipo 3: Quicksort, sección 7.4.2 del libro de texto.
4. Equipo 4: Encontrando la mediana, sección 7.5 del libro de texto.
5. Equipo 5: Multiplicación de matrices, sección 7.6 del libro de texto.
6. Equipo 6: Exponenciación, sección 7.7 del libro de texto.