Teoria de Algoritmos

Capitulo 2: Algoritmos Divide y Venceras

Tema 6: Aplicaciones

- Otras aplicaciones
  - Multiplicación de enteros
  - Multiplicación de matrices. Metodo de Strassen. Multiplicación de polinomios
  - El problema del “skyline”

Multiplicacion de enteros

- El problema que consideramos es el de la multiplicacion de enteros muy grandes.
- Sean x e y dos numeros de n bits.
- Los metodos tradicionales de multiplicacion requieren $O(n^2)$ operaciones sobre los bits:

```
x = 1 0 1 1 0 1 0 1
y = 1 0 1 0 0 1 1 0
-------------------
      1 0 1 1 0 1 0 1
    1 0 1 1 0 1 0 1
  1 0 1 1 0 1 0 1
1 0 1 1 0 1 0 1
-------------------
1 1 1 0 1 0 1 0 1 0 1 1 1 1 0
```

```
x = 1 0 1 1
y = 0 1 1 0
-----------
    1 0 1 1
  1 0 1 1
-----------
  1 0 0 0 0 1 0
```

Multiplicacion de enteros

- Supongamos que $n=2^m$,
- Tratamos a x e y como dos strings de n bits y los partimos en dos mitades como x = a * b, y= c * d, de manera que las concatenaciones de a, b y de c, d reproducen x e y.
- Como son numeros binarios, podemos expresarlos como

$$xy = (a2^{n/2} + b)(c2^{n/2} + d)$$
$$= ac2^n + (ad + bc)\,2^{n/2} + bd \text{ ..... } (*)$$

Ejemplo: Si x = 13 → x = 1101 (n = 4) → A = 3 y B = 1 → $13 = A \cdot 2^{n/2} + B = 3 \cdot 2^{n/2} + 1 = = (11)_{(2)} \cdot 2^{n/2} + (01)_{(2)}$.

- De esta manera el calculo de xy se simplifica a la realizacion de cuatro multiplicaciones de numeros de (n/2)-bits y algunas adiciones y desplazamientos de 0 y 1 (las multiplicaciones son por potencias de 2).

Multiplicacion de enteros

- Sea T(n) el numero de operaciones requeridas para multiplicar los dos numeros de n bits.
- Esta claro que,

$$
\begin{aligned}
T(n) &= 4T(n/2) + cn \\
&= 4\,(4T(n/2^2) + cn/2) + cn \\
&= 4^2T(n/2^2) + 2cn + cn \\
&= 4^3T(n/2^3) + 2^2cn + 2cn + cn \\
&\quad\ \mathbf{M} \\
&= 4^mT(1) + (2^{m-1} + \ldots + 1)\,cn \\
&= 2^m2^m + (2^m - 1)\,cn \\
&= O(n^2)
\end{aligned}
$$

- Con lo que se demuestra que no hemos adelantado nada

Multiplicacion de enteros

Pero, consideremos la siguiente recurrencia

$$T(n) = \begin{cases} b & \text{si } n = 1 \\ aT(n/c) + bn & \text{si } n > 1 \end{cases}$$

tomando $n = c^m$ podemos resolverla y obtener ($r = a/c$)

$$a < c\ ;\ r < 1,\ r^m \to 0,\ T(n) \to \frac{bc}{c-a} n = O(n)$$

$$a = c\ ;\ r = 1,\ T(n) = bn(m+1) = O(n\log n)$$

$$a > c\ ;\ r > 1,\ T(n) \to bnr^m = bn\left(\frac{a}{c}\right)^{\log_c n} = ba^{\log_c n} = O(n^{\log_c a})$$

- T(n) depende del numero de subproblemas y de su tamaño.
- Si el numero de subproblemas fuera 1, 3, 4, 8, entonces los algoritmos serian de ordenes n, $n^{\log 3}$, n^2, n^3 respectivamente.

Multiplicacion de enteros

- Si observamos la situacion, en nuestro caso, la multiplicacion de enteros va a producir algoritmos O(n^2) si el numero de subproblemas de tamaño mitad es cuatro, ya que el 4 que acompaña al t (n/2) es el que nos hace el orden cuadrático,
- Po tanto intentamos disminuir el número de subproblemas DV (concretamente con tres productos en vez de 4).
- Para reducir la complejidad en tiempo intentaremos reducir el numero de subproblemas (y por tanto de multiplicaciones) a costa de hacer mas adiciones y multiplicaciones por potencias de dos, y haremos asi una multiplicacion menos
- El truco es especialmente significativo conforme mas grande es n, es decir, asintoticamente

Multiplicacion de enteros

- Teniamos

$$xy = (a2^{n/2} + b)(c2^{n/2} + d)$$

- Ahora

$$x \cdot y = ac \cdot 2^n + [(a - b)(d - c) + ac + bd] \cdot 2^{n/2} + bd$$

- Es evidente que con esta forma de multiplicar x e y solo tenemos que hacer
  - tres multiplicaciones de dos numeros de (n/2)-bits
  - seis adiciones y
  - multiplicaciones por potencias de 2
- Por tanto el correspondiente algoritmo DV debe ser mas eficiente

Multiplicacion de enteros

- Podemos usar la rutina de multiplicacion recursivamente para calcular los tres productos. Las adiciones y desplazamientos requieren un tiempo O(n). Asi, la complejidad en tiempo de la multiplicacion de dos enteros de n bits esta acotada superiormente por

$$T(n) = \begin{cases} pn^2 & \text{si } n \le n_o \\ 3T(n/2) + kn & \text{si } n \ge n_o \end{cases}$$

- Donde k es una constante que incorpora las adiciones y las transferencias de bits.
- Es evidente que el tiempo de este algoritmo es $O(n^{\log 3}) = O(n^{1.59})$.

Ejemplo

$$x = 1\ 0\ 1\ 1 \qquad a = 1\ 0 \qquad b = 1\ 1$$
$$y = 0\ 1\ 1\ 0 \qquad c = 0\ 1 \qquad d = 1\ 0$$

$$u = (a+b)(c+d) = (1\ 0\ 1)(1\ 1) = 1\ 1\ 1\ 1$$
$$v = ac = (1\ 0)(0\ 1) = 1\ 0$$
$$w = bd = (1\ 1)(1\ 0) = 1\ 1\ 0$$
$$z = u - v - w = 1\ 1\ 1$$
$$xy = v2^4 + z2^2 + w = 1\ 0\ 0\ 0\ 0\ 0 + 1\ 1\ 1\ 0\ 0 + 1\ 1\ 0 = 1\ 0\ 0\ 0\ 0\ 1\ 0$$

Este mismo enfoque puede emplearse para diseñar un algoritmo que multiplique dos polinomios de grado n

Multiplicacion de matrices

- Si tenemos dos matrices A y B cuadradas en donde A tiene el mismo número de filas que columnas de B, se trata de multiplicar A y B para obtener una nueva matriz C.
- La multiplicacion de matrices se realiza conforme a

$$C_{ij} = \sum_{k=1}^{n} A_{ik} \cdot B_{kj}$$

- Esta fórmula corresponde a la multiplicación normal de matrices, que consiste en tres bucles anidados, por lo que es $O(n^3)$.
- Para aplicar la técnica DV, vamos a proceder como con la multiplicacion de enteros, con la intencion de obtener un algoritmo mas (?) eficiente para multiplicar matrices.

Multiplicacion de matrices

La multiplicacion puede hacerse como sigue:

$$\underset{C}{\begin{pmatrix} r & s \\ t & u \end{pmatrix}} = \underset{A}{\begin{pmatrix} a & b \\ c & d \end{pmatrix}} \underset{B}{\begin{pmatrix} e & g \\ f & h \end{pmatrix}} = \begin{pmatrix} ae+bf & ag+bh \\ ce+df & cg+dh \end{pmatrix}$$

- Esta formulacion divide una matriz *n* x *n en matrices de tamaños* n/2 x n/2, con lo que divide el problema en 8 subproblemas de tamaños n/2.
- Notese que n se usa como tamaño del caso aunque la dimension de la matriz es n^2
- Este enfoque da la siguiente recurrencia,

Multiplicacion de matrices

$$T(n) = \begin{cases} b & \text{si } n = 1 \\ 8T(n/2) + bn^2 & \text{si } n > 1 \end{cases}$$

- A partir de la cual, es evidente que T(n) sigue siendo O(n^3).
- Pero, basandonos en el enfoque DV que empleamos para multiplicar enteros, la multiplicacion de las matrices tambien puede calcularse como sigue.

El metodo de Strassen

$$\begin{pmatrix} r & s \\ t & u \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} e & g \\ f & h \end{pmatrix} = \begin{pmatrix} ae+bf & ag+bh \\ ce+df & cg+dh \end{pmatrix}$$

$\Uparrow$ C $\quad$ $\Uparrow$ A $\quad$ $\Uparrow$ B

$$\begin{aligned} P &= (a+d)(e+h) \\ Q &= (c+d)e \\ R &= a(g-h) \\ S &= d(f-e) \\ T &= (a+b)h \\ U &= (c-a)(e+g) \\ V &= (b-d)(f+h) \end{aligned}$$

$$\begin{aligned} r &= P+S-T+V \\ s &= R+T \\ t &= Q+S \\ u &= P+R-Q+U \end{aligned}$$

- Es evidente que solo se necesitan 7 multiplicaciones y 18 adiciones/substracciones, en lugar de las anteriores 8.

El metodo de Strassen

$$
\begin{aligned}
T(n) &= 7T(n/2) + bn^2 \\
&= 7\,(7T(n/4) + b(n/2)^2) + bn^2 \\
&= 7^2\,(7T(n/8) + b(n/4)^2) + (7/4 + 1)bn^2 \\
&= 7^m T(1) + ((7/4)^{m-1} + \ldots + (7/4) + 1)bn^2 \\
&= ((7/4)^m + \ldots + (7/4) + 1)bn^2 \\
&= ((7/4)^m - 1)bn^2 / ((7/4) - 1) \\
&= O(7^m) = O(n^{\log 7}) = O(n^{2.81})
\end{aligned}
$$

Se conocen mejoras del algoritmo, pero en la practica las rebajas que consiguen son a costa de grandes aumentos en los valores de las correspondientes constantes ocultas

¿Y si las matrices no fueran cuadradas ?

Multiplicacion de polinomios

- Supongamos dos polinomios:

$$P(x) = p_0 + p_1 x + p_2 x^2 + \ldots + p_{n-1} x^{n-1}$$

$$Q(x) = q_0 + q_1 x + q_2 x^2 + \ldots + q_{n-1} x^{n-1}$$

- Con el exponente n par: hay n coeficientes y el grado de cada polinomio es n – 1.
- DV sugiere dividir los polinomios por la mitad, quedando de la siguiente manera:
- P(x):

$$P_I(x) = p_0 + p_1 x + p_2 x^2 + \ldots + p_{n/2-1} x^{n/2-1}$$

$$P_D(x) = p_{n/2} + p_{n/2+1} x + p_{n/2+2} x^2 + \ldots + p_{n-1} x^{n/2-1}$$

- Q(x) (análogo a P (x)): $Q_I(x)$ y $Q_D(x)$

Multiplicacion de polinomios

- Entonces:

$$P(x) = P_I(x) + P_D(x)\, x^{n/2}$$

$$Q(x) = Q_I(x) + Q_D(x)\, x^{n/2}$$

- El algoritmo DV nos llevaría a:

$$P(x)\cdot Q(x) =$$

$$P_I(x)Q_I(x) + \left(P_I(x)Q_D(x) + P_D(x)Q_I(x)\right) x^{n/2} + P_D(x)Q_D(x)\, x^n$$

- El algoritmo subyacente es de orden cuadrado, como el convencional que se desprende de la multiplicacion de polinomios

Multiplicacion de polinomios

- Con el mismo metodo que en los problemas anteriores, la multiplicacion de los polinomios puede hacerse teniendo en cuenta que

$$R_I(x) = P_I(x)\, Q_I(x)$$
$$R_D(x) = P_D(x)\, Q_D(x)$$
$$R_H(x) = (P_I(x) + P_D(x))\,(Q_I(x) + Q_D(x))$$

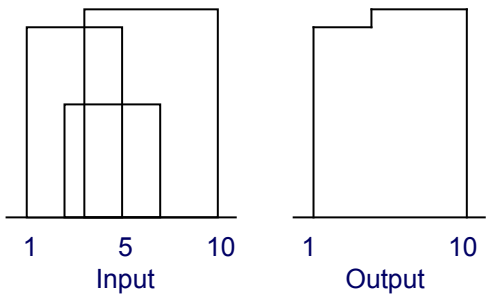
- Entonces la multiplicación de P por Q queda:
- $P(x)\cdot Q(x) = R_I(x) + (R_H(x) - R_I(x) - R_D(x))\, x^{n/2} + R_D(x)\, x^n$
- Que conlleva “solo” tres multiplicaciones de polinomios de grado mitad que los orginales

El problema de la linea del horizonte

- Con la comerzialización y popularización de las estaciones de trabajo graficas de alta velocidad, el CAD (“computer-aided design”) y otras areas (CAM, diseño VLSI) hacen un uso masivo y efectivo de los computadores.
- Un importante problema a la hora de dibujar imágenes con computadores es la eliminación de lineas ocultas, es decir, la eliminación de lineas que quedan ocultadas por otras partes del dibujo.
- Este interesante problema recibe el nombre de Problema de la Linea del Horizonte (“Skyline Problem”).

El problema de la linea del horizonte

- El problema se establece en los siguientes sencillos terminos
- Dadas las situaciones exactas (coordenadas)
  - Por ejemplo (1,11,5), (2.5,6,7), (2.8,13,10), donde cada valor es la coordenada izquierda, la altura y la coordenada derecha de un edificio
- y las formas de n edificios rectangulares en una ciudad bi-dimensional,



- construir un algoritmo Divide y Vencerás que calcule eficientemente el “skyline” (en dos dimensiones) de esos edificios, eliminando las lineas ocultas

Particularización del metodo DV

- **1. Tamaño:** el problema es una colección de edificios, por tanto el numero de edificios en una colección es una elección natural para la forma de medir el tamaño del problema, asi
  - Tamaño(Edificios) = numero de edificios en el input.
- **2. Caso base del problema:** Como el input esta restringido a la coleccion de edificios, entonces el caso base del problema es una colección constituida por un solo edificio, puesto que el “skyline” de un unico edificio es el mismo edificio. Por tanto
  - TamañoCasoBase = 1

Particularización del metodo DV

- **3. Solución del Caso Base**: Como la solucion para un unico edificio es el edificio en si mismo, el cuerpo del procedimiento

  EncuentraSolucionCasoBase(Edificios; Skyline)

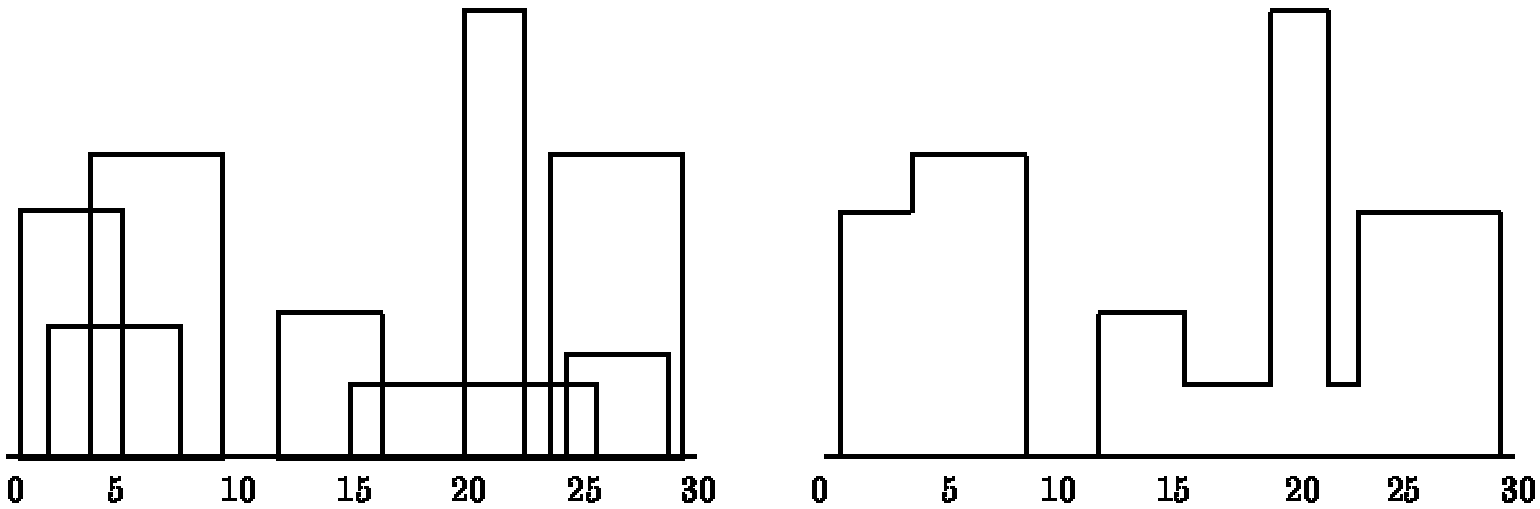
  puede definirse como sigue:

```
Procedimiento EncuentraSolucionCasoBase(Edificios; Skyline)
  Begin
      Skyline = Edificios[1]
  end;
```

Particularización del metodo DV

- **4. División del problema**: El unico requerimiento es que el tamaño de los subproblemas obtenidos, de la colección de edificios, debe ser estrictamente menor que el tamaño del problema original.
- Teniendo en cuenta la complejidad en tiempo del algoritmo, funcionará mejor si los tamaños de los subproblemas son parecidos.
- Diseñaremos un procedimiento de division de manera que los dos subproblemas que obtendremos, EdificiosIzquierda y EdificiosDerecha, tengan tamaños aproximadamente iguales
- **5. Combinación**: Hay que combinar dos skylines subsoluciones en un unico skyline:
- En esencia la parte de combinación del algoritmo lo que hace es tomar dos skylines, los analiza punto a punto desde la izquierda a la derecha, y en cada punto toma el mayor valor de los dos skylines.
- Ese punto es el que toma como valor del skyline combinado

Ejemplo



Input (skyline de la izquierda)

(1,11,5), (2,6,7), (3,13,9), (12,7,16), (14,3,25), (19,18,22), (23,13,29), (24,4,28)

Output (skyline de la derecha)

1 11 3 13 9 0 12 7 16 3 19 18 22 3 23 13 29 0

Algoritmo DV del Skyline

```
Procedimiento Skyline(Edificios,Skyline)
  Begin
    If CasoBase(Edificios)
      Then
        EncuentraSolucionCasoBase(Edificios,Skyline)
      Else
        Begin
          Dividir(Edificios, EdificiosIzquierda, EdificiosDerecha)
          EncuentraSkyline(EdificiosIzquierda, SkylineIzquierda)
          EncuentraSkyline(EdificiosDerecha, SkylineDerecha)
          Combina(SkylineIzquierda, SkylineDerecha, Skyline)
        End
End;
```

Algoritmo DV del Skyline

- Por tanto siguiendo la estrategia del DV, en ese algoritmo primero comprobamos si el array de edificios que nos dan contiene un solo edificio.
- Si es asi, la solucion es ese mismo edificio
- En caso contrario, dividimos el conjunto de edificios, resolvemos recursivamente cada mitad, y finalmente combinamos los dos skylines obtenidos
- El tiempo del algoritmo se encuentra a partir de la recurrencia

$$T(n) = 2T(n/2)+O(n)$$

- Por tanto el algoritmo tiene orden O(nlogn)