Tema 4: Divide y vencerás

Método general

1. **Dividir** el problema (P) en subproblemas (P_i) más fáciles de resolver (los subproblemas no tienen que ser disjuntos)
2. **Resolver** los subproblemas
3. **Combinar** las soluciones (S_i) para obtener la solución (S) del problema inicial

Es común utilizar **recursividad** para obtener soluciones eficientes para problemas que se pueden dividir en subproblemas reducidos y resolubles.

Determinación de umbral

Tenemos un algoritmo recursivo $t_c(n)$ que resuelve un problema dividiéndolo en 3 subproblemas de tamaño $n/2$ siendo cada uno de ellos del tipo $t_A(n)$. Entonces se define:

$$t_c(n) \begin{cases} = t_A(n)\ \forall n \leq n_0 \\ = t_C(n/2) + t(n)\ \forall n > n_0 \end{cases}$$

El valor n_0 es el **umbral**

Cada implementación tiene un único umbral

Análisis algoritmos

Nos encontraremos recurrencias del tipo:

$$T(n) \begin{cases} = t(n), \text{ si } n \leq c \\ = aT(n/b) + D(n) + C(n) \text{ en otro caso} \end{cases}$$

a es el número de subproblemas
n/b el tamaño de cada subproblema
$D(n)$ el tiempo de dividir los problemas en subproblemas
$C(n)$ el tiempo de combinar las soluciones

Tema 5: Búsqueda y Ordenación

Búsqueda binaria

Se va dividiendo el vector en mitades y se busca en la mitad en la que se encuentre. Primero habrá que ordenar el vector.

$$T(n) \begin{cases} = O(1)\ \forall n \leq 0 \\ = T(n/2) + a\ \forall n \geq 0 \end{cases}$$

Tenemos que, $T(n) = c_1 log(n)$, luego $O(log(n))$

Algoritmo general ordenación

Iniciar Ordenar(lista)
    Partir la lista en dos listas (izquierda y derecha)
    Iniciar Ordenar(izquierda)
    Inicial Ordenar(derecha)
    Combinar izquierda y derecha
End

Ordenación por mezcla (mergeSort)

Nos dan un k
Iniciar Ordenar(lista)
    Partir lista en A (con n/k) elementos y B (resto de elementos)
    Iniciar Ordenar(A)
    Iniciar Ordenar(B)
    Combinar A y B usando mezcla
End

Para $k = 2$, tenemos que $T(n) = 2^i T(n/2^i) + ic_2 n$ que se resuelve con $T(n) = c_1 n + c_2 n\, log(n)$. Luego, $O(n\, log(n))$

QuickSort (Algoritmo de Hoare)

Iniciar QuickSort(T[i..j])
    Si j-i es pequeño, entonces Inserción T([i..j]) **
    Pivoteo (Pivote elemento l)
    Iniciar QuickSort(T[i..l-1])
    Iniciar QuickSort(T[l…j])
End

** concretamente, más pequeño que un umbral que se determinará teórica o empíricamente

- Sobre el **pivoteo**
  La función Pivoteo() de arriba, pone a la izquierda del pivote los números menores que él, y a su derecha, los mayores
  Para la elección del pivote se suele usar la mediana del array . También podemos elegir tres elementos al azar y escoger su mediana (suele reducir el tiempo de ejecución 5%)
  La mejor forma de pivotear es escoger como pivote, entre los dos primeros elementos del array, el mayor de ellos.
- Eficiencia:
  - Tenemos que $T(n) = 2n\, log(n) + O(n)$, luego tiempo promedio $O(n\, log(n))$
  - En el peor caso, $O(n^2)$
- En la práctica, es el mejor algoritmo de ordenación

Tema 6: Aplicaciones

Multiplicación de enteros

Iniciar Mult(a,b)

$n = max\{a, b\}$

aL y aR las mitades izquierda y derecha de a

bL y bR las mitades izquierda y derecha de b

$x_1 =$ Mult(aL, bL)

$x_2 =$Mult(aL, bR)

$x_3 =$Mult(aR, bL)

$x_4 =$Mult(aR, bR)

devolver $x_1 10^n + (x_2 + x_3)10^{n/2} + x_4$

End

Hacer $(x_2 + x_3)$ es lo mismo que hacer $(aL + aR)(bL + bR) - x_1 - x_4$. De esta forma, el algoritmo realizaría 3 multiplicaciones, 6 adiciones y dos multiplicaciones por potencias.

Cuando se hace en base 10, se conoce como el **Algoritmo de Karatsuba**

//la profe lo hace en base 2

$$T(n) = pn^2 \ \forall n \leq n_0$$
$$= 3T(n/2) + kn \ \forall n \geq n_0$$

donde k es una constante que incorpora las adiciones y transferencias de bits

Luego, la eficiencia es $O(n^{log(3)})$

Multiplicación de matrices cuadradas

La multiplicación normal de matrices consiste en 3 bucles anidados, luego $O(n^3)$

Con Divide y Vencerás:

$$T(n) = b, \ si \ \ n = 1$$
$$= 8T(n/2) + bn^2, \ \ si \ \ n > 1$$

Luego, la eficiencia sigue siendo $O(n^3)$

Nota: la eficiencia de cualquier algoritmo de multiplicación de matrices no puede bajar de $O(n^2)$

Método de Strassen

Para multiplicar matrices de forma eficiente.

Divide las matrices en submatrices y, recursivamente, sigue dividiendo las submatrices en otras submatrices. Y las multiplica recursivamente.

$$T(n) = 7T(n/2) + bn^2 \Rightarrow O(n^{log(7)})$$

- El 7 que multiplica a $T(n/2)$ viene de hacer un procedimiento similar al de la multiplicación de enteros, para que solo necesitemos 7 multiplicaciones, en lugar de 8.
- Hay mejores algoritmos, pero aumentan mucho las constantes ocultas

Multiplicación de polinomios

Dados dos polinomios: $P(x) = p_0 + p_1 x + \ldots + p_{n-1} x^{n-1}$

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

DV sugiere dividir los polinomios por la mitad:

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

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

De forma que: $P(x) = P_I(x) + P_D(x) x^{n/2}$

$$Q(x) = Q_1(x) + Q_D(x) x^{n/2}$$

Como en los problemas anteriores, podemos hacer la multiplicación de la siguiente forma:

$P(x)Q(x) = R_I(x) + (R_H(x) - R_I(x) - R_D(x)) x^{n/2} + R_D(x) x^n$

Donde: $R_I(x) = P_I(x) Q_I(x)$

$R_D(x) = P_D(x) Q_D(x)$

$R_H(x) = (P_I(x) + P_D(x))(Q_I(x) + Q_D(x))$

Donde conseguimos tres multiplicaciones de polinomios de grado mitad que los originales, luego la eficiencia ya es menor que $O(n^2)$

El problema de skyline

Se dan coordenadas de la forma (coordenada izquierda, altura, coordenada derecha) de edificios rectangulares en una ciudad bidimensional.

El problema consiste en calcular el skyline de los edificios.

- Particularización:
  - Tamaño: número edificios
  - Tamaño caso base = 1
  - Solución del caso base: SkyLine=Edificios[1]
  - División del problema: dividir en EdificiosIzquierda y EdificiosDerecha
  - Combinación: para combinar dos skylines, toma punto a punto desde la izquierda a la derecha, y en cada punto toma el mayor valor de los dos skylines

El tiempo del algoritmo $T(n) = 2T(n/2) + O(n)$, por tanto el orden es $O(n\, log(n))$