Dividir y Conquistar

Dividir y Conquistar

- Conocida como
  - "Divide & Conquer"
  - "Dividir y Conquistar
  - "Divide y Reinarás
  - D&C
  - etc.
- La metodología consiste en
  - dividir un problema en problemas similares....pero más chicos
  - resolver los problemas menores
  - Combinar las soluciones de los problemas menores para obtener la solución del problema original.
- El método tiene sentido siempre y cuando la división y la combinación no sean excesivamente caras
- ¿Entonces?

Esquema general de D&C

- Algoritmo DC(X)
  - Si X es suficientemente chico (o simple)
    - ADHOC(X)
  - En caso contrario,
    - Descomponer X en subinstancias X_1, X_2,.... X_k
    - Para i desde 1 hasta k hacer
      - $Y_i \leftarrow DC(X_i)$
    - Combinar las soluciones Y_i para construir una solución Y para X

Ejemplo: Algoritmo de Strassen

- Queremos multiplicar dos enteros de *n* cifras en base *b*, *X* e *Y*
- Algoritmo tradicional requiere....$O(n^2)$
- Pero...
  - Sea $X = X_1\, b^{(n/2)} + X_0$, e $Y = Y_1\, b^{(n/2)} + Y_0$
- Entonces,
  - $XY = X_1\, Y_1\, b^n + (X_0\, Y_1 + X_1\, Y_0)\, b^{n/2} + X_0\, Y_0$
- Si calculamos la complejidad de este método, nos da $O(n^2)$ (ver después)

Ejemplo: Algoritmo de Strassen (II)

- Pero si definimos
  - $m_1 = X_0 Y_0$, $m_2 = X_1 Y_1$ y $m_3 = (X_0 - X_1)(Y_1 - Y_0)$
- Resulta que
  - $XY = m_2\, b^n + (m_1 + m_2 + m_3)\, b^{n/2} + m_1$
- O sea tenemos 3 subproblemas de tamaño n/2, lo que mejora la complejidad total del algoritmo llevándola a $T(n)=O(n^{\log_2 3}) \sim O(n^{1.59})$ (ver después)
- Strassen aplicó esta misma idea a la multiplicación de matrices (¡pensarla!)

Recurrencias típicas de D&C

- Esquema D&C: dividir las instancias de tamaño mayor que cierto n_0 en sub-instancias, resolverlas y luego combinar las soluciones para el problema inicial.
- Situación típica: dividir en a subproblemas, de tamaño máximo n/c, el costo de determinar subproblemas (divide) y unirlos (conquer) es bn^d. Suponemos la base ($n=1$) cuesta b.

Solución de la recurrencia típica

- Costo (suponiendo $n=c^k$)
  - $T(n)=aT(n/c)+bn^d = aT(c^{k-1})+bn^d=$
  - $= a(aT(c^{k-2})+(bc^{d(k-1)}))+ bc^{kd}= a^2T(c^{k-2})+abc^{d(k-1)}+ bc^{kd}=$
  - $= a^2(a(T(c^{k-3})+b(c^{k-2})^d)+abc^{d(k-1)}+ bc^{kd} =$
  - $= a^3T(c^{k-3})+a^2b(c^{k-2})^d+abc^{d(k-1)}+ bc^{kd} =$
  - $= a^3T(c^{k-3})+a^2bc^{d(k-2)}+abc^{d(k-1)}+ bc^{kd} =$
  - =..............................=
  - $a^jT(c^{k-j})+ \sum_{i=0}^{j-1} a^ibc^{d(k-i)}=$
  - =...........................= ¿hasta cuándo? Hasta $c^{(k-j)}=1$ o sea $j=\log_c n$
  - $= a^kb+ b\sum_{i=0}^{k-1} a^ic^{d(k-i)}= b\sum_{i=0}^{k} a^ic^{d(k-i)}=$
  - $bc^{dk}\sum_{i=0}^{k} a^ic^{-di}= bn^d\sum_{i=0}^{\log n} (a/c^d)^i = ?$

Análisis por casos

- a=1, d=0 (o sea, un solo subproblema, la combinación tiene costo constante)

$$b\sum^{\log_c n} 1 = O(\log_c n)$$

- d=1 (o sea "conquer lineal")
  - Caso a<c (o sea, "pocos subproblemas")
  - Como a/c<1, la serie converge →

$$bn\sum_{i=0}^{\log_c n} (a/c)^i = bn.const = O(n)$$

Análisis por casos

- d=1 (o sea "conquer lineal")
  - Caso a=c

$$bn\sum^{\log_c n} 1 = O(n\log_c n)$$

  - Caso a>c (o sea, muchos subproblemas)

$$T(n) = bn\sum_{i=0}^{\log_c n}(a/c)^i = bn\frac{\left(\frac{a}{c}\right)^{\log_c n+1}-1}{\frac{a}{c}-1} = O(n\left(\frac{a}{c}\right)^{\log_c n}) =$$

$$O(n\frac{a^{\log_c n}}{c^{\log_c n}}) = O(n\frac{a^{\log_c n}}{n}) = O(a^{\log_a n \times \log_c a}) = O(n^{\log_c a})$$

Teorema Maestro (Master theorem)

- Permite resolver relaciones de recurrencia de la forma:

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

- La solución es

$$T(n) = \Theta(n^{\log_c a}) \text{ si } \quad f(n) = O(n^{\log_c a - \varepsilon}) \text{ para } \varepsilon > 0$$

$$T(n) = \Theta(n^{\log_c a} \log n) \text{ si } \quad f(n) = \Theta(n^{\log_c a})$$

$$T(n) = \Theta(f(n)) \text{ si } \quad \begin{array}{l} f(n) = \Omega(n^{\log_c a + \varepsilon}) \text{ para } \varepsilon > 0 \text{ y} \\ af(n/c) < kf(n) \text{ para } k < 1 \text{ y } n \text{ suficientemente grande} \end{array}$$