Algoritmos y Estructuras de Datos II

Recurrencias Divide y Vencerás

30 de marzo de 2015

Contenidos

Algoritmos de ordenación

- Algoritmos elementales:
  - Ordenación por selección
  - Ordenación por inserción
- Algoritmos eficientes:
  - Ordenación por intercalación
  - Ordenación rápida

Número de comparaciones

- Ordenación por selección: $\mathcal{O}(n^2)$ (cuadrático).
- Ordenación por inserción:
  - peor caso (arreglo invertido): $\mathcal{O}(n^2)$.
  - mejor caso (arreglo ordenado): $\mathcal{O}(n)$ (lineal).
  - caso medio: $\mathcal{O}(n^2)$.
- Ordenación por intercalación: $\mathcal{O}(n \log n)$ ($\subset \mathcal{O}(n^{1.0001})$).
- Ordenación rápida:
  - peor caso (arreglo ordenado o invertido): $\mathcal{O}(n^2)$.
  - mejor caso (pivots parten en mitades): $\mathcal{O}(n \log n)$.
  - caso medio: $\mathcal{O}(n \log n)$.

Jerarquía

$$\mathcal{O}(1) \subset \mathcal{O}(\log_2 n) = \mathcal{O}(\log_3 n) \subset \mathcal{O}(n^{0.001}) \subset \mathcal{O}(n^{1.5}) \subset \mathcal{O}(n^2) \subset$$

$$\subset \mathcal{O}(n^5) \subset \mathcal{O}(n^{100}) \subset \mathcal{O}(1.01^n) \subset \mathcal{O}(2^n) \subset \mathcal{O}(100^n) \subset$$

$$\subset \mathcal{O}(10000^n) \subset \mathcal{O}(n!) \subset \mathcal{O}(n^n)$$

Propiedades

- Constantes multiplicativas no afectan.
- Términos de crecimiento despreciable no afectan.
- Sean $a, b > 1$, $\mathcal{O}(\log_a n) = \mathcal{O}(\log_b n)$.
- Regla del límite. Jerarquía.
- Sea $\forall^\infty n \in \mathbb{N}.f(n) > 0$. Entonces
  $g(n) \in \mathcal{O}(h(n)) \iff f(n)g(n) \in \mathcal{O}(f(n)h(n))$.
- Sea $\lim_{n\to\infty} h(n) = \infty$. Entonces
  $f(n) \in \mathcal{O}(g(n)) \implies f(h(n)) \in \mathcal{O}(g(h(n)))$.

Jerarquía

$$\mathcal{O}(1) \subset \mathcal{O}(\log(\log(\log n))) \subset \mathcal{O}(\log(\log n)) \subset \mathcal{O}(\log n) \subset \mathcal{O}(n^{0.001}) \subset$$

$$\subset \mathcal{O}(n) \subset \mathcal{O}(n \log n) \subset \mathcal{O}(n^{1.001}) \subset \mathcal{O}(n^{100}) \subset \mathcal{O}(1.01^n) \subset$$

$$\subset \mathcal{O}(n^{100} * 1.01^n) \subset \mathcal{O}(1.02^n) \subset \mathcal{O}(100^n) \subset \mathcal{O}(10000^n) \subset$$

$$\subset \mathcal{O}((n-1)!) \subset \mathcal{O}(n!) \subset \mathcal{O}((n+1)!) \subset \mathcal{O}(n^n)$$

Recurrencias

- Surgen al analizar algoritmos recursivos, como la ordenación por intercalación.
- El conteo de operaciones “copia” la recursión del algoritmo y se vuelve recursivo también.
- Ejemplo: conteo de comparaciones de la ordenación por intercalación.
- Es un ejemplo de algoritmo divide y vencerás.
- Es un ejemplo de recurrencia divide y vencerás:

$$t(n) = \begin{cases} 0 & \text{si } n \in \{0, 1\} \\ t(\lceil n/2 \rceil) + t(\lfloor n/2 \rfloor) + n - 1 & \text{si } n > 1 \end{cases}$$

Algoritmo divide y vencerás
Características

- hay una solución para los casos sencillos,
- para los complejos, se **divide** o **descompone** el problema en subproblemas:
  - cada subproblema es de igual naturaleza que el original,
  - el tamaño del subproblema es una **fracción** del original,
  - se resuelven los subproblemas apelando al mismo algoritmo,
- se **combinan** esas soluciones para obtener una solución del original.

Algoritmo divide y vencerás

Forma general

fun DV(x) **ret** y
    if x suficientemente pequeño o simple **then** y:= ad_hoc(x)
    else descomponer x en $x_1, x_2, \ldots, x_a$
        for i:= 1 **to** a **do** y_i:= DV(x_i) **od**
        combinar $y_1, y_2, \ldots, y_a$ para obtener la solución y de x
    fi
end

Normalmente los x_i son **fracciones** de x:

$$|x_i| = \frac{|x|}{b}$$

para algún b fijo.

Algoritmo divide y vencerás
Ejemplos

- Ordenación por intercalación:
  - "x simple" = fragmento de arreglo de longitud 0 ó 1
  - "descomponer" = partir al medio ($b = 2$)
  - $a = 2$
  - "combinar" = intercalar
- Ordenación rápida:
  - "x simple" = fragmento de arreglo de longitud 0 ó 1
  - "descomponer" = separar los menores de los mayores ($b = 2$)
  - $a = 2$
  - "combinar" = yuxtaponer

Algoritmo divide y vencerás
Conteo

Si queremos contar el costo computacional (número de operaciones) $t(n)$ de la función *DV* obtenemos:

$$t(n) = \begin{cases} c & \text{si la entrada es pequeña o simple} \\ a * t(n/b) + g(n) & \text{en caso contrario} \end{cases}$$

si c es una constante que representa el costo computacional de la función ad_hoc y $g(n)$ es el costo computacional de los procesos de descomposición y de combinación.
Esta definición de $t(n)$ es recursiva (como el algoritmo *DV*), se llama **recurrencia**. Existen distintos tipos de recurrencia. Ésta se llama **recurrencia divide y vencerás**.

Recurrencias divide y vencerás

Por la forma de la recurrencia

$$t(n) = \begin{cases} c & \text{si la entrada es pequeña o simple} \\ a * t(n/b) + g(n) & \text{en caso contrario} \end{cases}$$

resulta más sencillo calcular $t(n)$ cuando n es potencia de b.
Se organiza la tarea en dos partes

- calcular el orden de $t(n)$ cuando n es potencia de b,
- extender el cálculo para los demás n.

Extensión de la notación $\mathcal{O}$

- Sea $g : \mathbb{N} \overset{\infty}{\to} \mathbb{R}^{\geq 0}$,
  - $\mathcal{O}(g(n)|n \text{ potencia de } b) =$
    $= \{f : \mathbb{N} \overset{\infty}{\to} \mathbb{R}^{\geq 0} \mid \exists c > 0.\ \forall^{\infty} k \in \mathbb{N}. f(b^k) \leq c g(b^k)\}$
  - $\Omega(g(n)|n \text{ potencia de } b) =$
    $= \{f : \mathbb{N} \overset{\infty}{\to} \mathbb{R}^{\geq 0} \mid \exists c > 0.\ \forall^{\infty} k \in \mathbb{N}. f(b^k) \geq c g(b^k)\}$
  - $\Theta(g(n)|n \text{ potencia de } b) =$
    $\mathcal{O}(g(n)|n \text{ potencia de } b) \cap \Omega(g(n)|n \text{ potencia de } b)$

Calculando para n potencia de b

- Supongamos que
  - $$t(n) = \begin{cases} c & \text{si la entrada es pequeña o simple} \\ a * t(n/b) + g(n) & \text{en caso contrario} \end{cases}$$
  - y $g(n) \in \mathcal{O}(n^k)$, es decir $g(n) \leq dn^k$ para casi todo $n \in \mathbb{N}$
- n potencia de b, $n = b^m$
- $$\begin{aligned} t(n) &= t(b^m) \\ &= a * t(b^m/b) + g(b^m) \\ &\leq a * t(b^{m-1}) + d(b^m)^k \\ &\leq a * t(b^{m-1}) + d(b^k)^m \end{aligned}$$

Iterando

$$\begin{aligned}
t(b^m) &\leq at(b^{m-1}) + d(b^k)^m \\
&\leq a(t(b^{m-2}) + d(b^k)^{m-1}) + d(b^k)^m \\
&\leq a^2 t(b^{m-2}) + ad(b^k)^{m-1} + d(b^k)^m \\
&\leq a^3 t(b^{m-3}) + a^2 d(b^k)^{m-2} + ad(b^k)^{m-1} + d(b^k)^m \\
&\leq \ldots \\
&\leq a^m t(1) + a^{m-1} db^k + \ldots + ad(b^k)^{m-1} + d(b^k)^m \\
&= a^m c + d(b^k)^m((a/b^k)^{m-1} + \ldots + a/b^k + 1) \\
&= a^m c + d(b^m)^k(r^{m-1} + \ldots + r + 1)
\end{aligned}$$

donde $r = a/b^k$

$$\begin{aligned}
t(n) &\leq a^{\log_b n} c + dn^k(r^{m-1} + \ldots + r + 1) \\
&= n^{\log_b a} c + dn^k(r^{m-1} + \ldots + r + 1)
\end{aligned}$$

Propiedad del logaritmo

En el último paso hemos usado que $x^{\log_y z}$ es igual a $z^{\log_y x}$.

- En efecto, si aplicamos $\log_y$ a ambos, obtenemos
- $\log_y(x^{\log_y z})$ y $\log_y(z^{\log_y x})$, que luego de simplificar quedan
- $(\log_y x)(\log_y z)$ y $(\log_y z)(\log_y x)$ que son iguales.
- Como $\log_y$ es inyectiva, $x^{\log_y z} = z^{\log_y x}$ vale.

Volvamos a los cálculos.

Finalizando

$$t(n) \leq n^{\log_b a} c + dn^k (r^{m-1} + \ldots + r + 1)$$

donde $r = a/b^k$

- si $r = 1$, entonces $a = b^k$ y $\log_b a = k$ y además

$$\begin{aligned} t(n) &\leq n^{\log_b a} c + dn^k m \\ &= n^k c + dn^k \log_b n \\ &\in \mathcal{O}(n^k \log n | n \text{ potencia de } b) \end{aligned}$$

- si $r \neq 1$, entonces

$$t(n) \leq n^{\log_b a} c + dn^k \left(\frac{r^m - 1}{r - 1}\right)$$

Finalizando
caso $r \neq 1$

- si $r > 1$, como $r = a/b^k$ entonces $a > b^k$ y $\log_b a > k$ y además

$$\begin{aligned}
t(n) &\leq n^{\log_b a} c + dn^k \left(\frac{r^m - 1}{r-1}\right) \\
&\leq n^{\log_b a} c + \frac{d}{r-1} n^k r^m \\
&= n^{\log_b a} c + \frac{d}{r-1} n^k \frac{a^m}{(b^k)^m} \\
&= n^{\log_b a} c + \frac{d}{r-1} n^k \frac{a^m}{(b^m)^k} \\
&= n^{\log_b a} c + \frac{d}{r-1} n^k \frac{a^m}{n^k} \\
&= n^{\log_b a} c + \frac{d}{r-1} a^m \\
&= n^{\log_b a} c + \frac{d}{r-1} a^{\log_b n} \\
&= n^{\log_b a} c + \frac{d}{r-1} n^{\log_b a} \\
&\in \mathcal{O}(n^{\log_b a} | n \text{ potencia de } b)
\end{aligned}$$

Finalizando
caso $r < 1$

- si $r < 1$, como $r = a/b^k$ entonces $a < b^k$ y $\log_b a < k$. Además, $r - 1$ y $r^m - 1$ son negativos, para evitar confusión escribimos $\frac{1-r^m}{1-r}$ en vez de $\frac{r^m-1}{r-1}$.

$$\begin{aligned} t(n) &\leq n^{\log_b a} c + dn^k \left(\tfrac{1-r^m}{1-r}\right) \\ &= n^{\log_b a} c + \tfrac{d}{1-r} n^k (1 - r^m) \\ &\leq n^{\log_b a} c + \tfrac{d}{1-r} n^k \\ &\in \mathcal{O}(n^k | n \text{ potencia de } b) \end{aligned}$$

Conclusión

- Para
  - $$t(n) = \begin{cases} c & \text{si la entrada es pequeña o simple} \\ a * t(n/b) + g(n) & \text{en caso contrario} \end{cases}$$
  - con $g(n) \in \mathcal{O}(n^k)$,
  - demostramos
$$t(n) \in \begin{cases} \mathcal{O}(n^{\log_b a} | n \text{ potencia de } b) & \text{si } a > b^k \\ \mathcal{O}(n^k \log n | n \text{ potencia de } b) & \text{si } a = b^k \\ \mathcal{O}(n^k | n \text{ potencia de } b) & \text{si } a < b^k \end{cases}$$
- Similar resultado vale reemplazando $\mathcal{O}$ por Ω en las 4 ocurrencias.
- Similar resultado vale reemplazando $\mathcal{O}$ por Θ en las 4 ocurrencias.

Extendiendo el resultado a todo n

- Hemos calculado el orden de cualquier algoritmo divide y vencerás, para n potencia de b.
- Queremos calcular el orden para todo n.
- Veremos que si $t(n)$ es no decreciente, y $t(n) \in \mathcal{O}(h(n)|n \text{ potencia de } b)$ para cualquiera de las tres funciones h que acabamos de considerar, entonces $t(n) \in \mathcal{O}(h(n))$.
- Similares resultados valen para Ω y Θ.

Funciones eventualmente no decrecientes

- Sea $f : \mathbb{N} \overset{\infty}{\to} \mathbb{R}^{\geq 0}$,
- se dice que f es **eventualmente no decreciente** si $\forall^{\infty} n \in \mathbb{N}, f(n) \leq f(n+1)$.
- Ejemplos:
  - la función $t(n)$ considerada al analizar la ordenación por intercalación.
  - las funciones n, n^2, etc., todas las funciones de la forma n^x con $x \geq 0$.
  - $\log_b n$ para $b > 1$.
  - La suma y el producto de funciones eventualmente no decrecientes, por ejemplo $n \log n$.

Funciones uniformes

- Sea $f : \mathbb{N} \overset{\infty}{\to} \mathbb{R}^{\geq 0}$ y sea $i \in \mathbb{N}^{\geq 2}$,
- se dice que f es i**-suave** o i**-uniforme** si
  - f es eventualmente no decreciente, y
  - $\exists d \in \mathbb{R}^+, \forall^\infty n \in \mathbb{N}, f(i * n) \leq d * f(n)$
- Ejemplos:
  - las funciones n, n^2, etc., todas las funciones de la forma n^x con $x \geq 0$.
  - $\log_b n$ para $b > 1$.
  - La suma y el producto de funciones i-uniforme, por ejemplo $n \log n$.
  - $n^{\log n}$, 2^n o $n!$ son ejemplos de funciones que no son i-uniformes, para ningún i.

Funciones uniformes

Lema

- Sea $f : \mathbb{N} \overset{\infty}{\to} \mathbb{R}^{\geq 0}$ y sean $i, j \in \mathbb{N}^{\geq 2}$,
- entonces, f es i-uniforme sii f es j-uniforme.
- Demostración:
  - Sea f i-uniforme.

$$
\begin{array}{rcll}
f(j * n) & = & f(i^{\log_i j} n) & j = i^{\log_i j} \\
& \leq & f(i^{\lceil \log_i j \rceil} n) & f \text{ es eventualmente no decreci} \\
& \leq & d f(i^{\lceil \log_i j \rceil - 1} n) & f \text{ es } i\text{-uniforme} \\
& \leq & d^2 f(i^{\lceil \log_i j \rceil - 2} n) & f \text{ es } i\text{-uniforme} \\
& & \ldots & \\
& \leq & d^{\lceil \log_i j \rceil} f(n) & f \text{ es } i\text{-uniforme}
\end{array}
$$

- Entonces a las funciones i-uniformes se las llama también **uniformes**.

Regla de uniformidad

- Sea $f : \mathbb{N} \overset{\infty}{\to} \mathbb{R}^{\geq 0}$ uniforme.
- Sea $t : \mathbb{N} \overset{\infty}{\to} R^{\geq 0}$ eventualmente no decreciente,
- si $t(n) \in \mathcal{O}(f(n)|n \text{ potencia de } b)$
- entonces $t(n) \in \mathcal{O}(f(n))$.
- Análogamente para Θ y Ω.
- Demostración: Sea n suficientemente grande, y sea m tal que $b^m \leq n < b^{m+1}$. Entonces

$$
\begin{array}{rcll}
t(n) & \leq & t(b^{m+1}) & t \text{ es eventualmente no decreciente} \\
 & \leq & cf(b^{m+1}) & t(n) \in \mathcal{O}(f(n)|n \text{ potencia de } b) \\
 & \leq & cdf(b^m) & f \text{ es } b\text{-uniforme} \\
 & \leq & cdf(n) & f \text{ es eventualmente no decreciente}
\end{array}
$$

por lo tanto $t(n) \in \mathcal{O}(f(n))$.

Recurrencias divide y vencerás

- Las funciones $n^{\log_b a}$, n^k y $n^k \log n$ son uniformes.
- Por lo tanto, para
  - $$t(n) = \begin{cases} c & \text{si la entrada es pequeña o simple} \\ a * t(n/b) + g(n) & \text{en caso contrario} \end{cases}$$

    si $t(n)$ es ev. no decreciente, y $g(n) \in \mathcal{O}(n^k)$, entonces
  - $$t(n) \in \begin{cases} \mathcal{O}(n^{\log_b a}) & \text{si } a > b^k \\ \mathcal{O}(n^k \log n) & \text{si } a = b^k \\ \mathcal{O}(n^k) & \text{si } a < b^k \end{cases}$$
- Similar resultado vale reemplazando $\mathcal{O}$ por Ω en las 4 ocurrencias.
- Similar resultado vale reemplazando $\mathcal{O}$ por Θ en las 4 ocurrencias.

Ejemplo: búsqueda binaria

{Pre: $1 \leq$ izq $\leq$ n+1 $\wedge$ $0 \leq$ der $\leq$ n $\wedge$ a ordenado}
fun binary_search_rec (a: **array**[1..n] **of T**, x:**T**, izq, der : **nat**) **ret** i:**na**
    var med: **int**
    if izq $>$ der $\rightarrow$ i = 0
      izq $\leq$ der $\rightarrow$ med:= (izq+der) $\div$ 2
          if x $<$ a[med] $\rightarrow$ i:= binary_search_rec(a, x, izq, med-1)
            x $=$ a[med] $\rightarrow$ i:= med
            x $>$ a[med] $\rightarrow$ i:= binary_search_rec(a, x, med+1,der)
          fi
    fi
end fun
{Post: (i $=$ 0 $\Rightarrow$ x no está en a[izq,der]) $\wedge$ (i $\neq$ 0 $\Rightarrow$ x = a[i])}

Búsqueda binaria
Función principal

{Pre: n $\geq$ 0 }
fun binary_search (a: **array**[1..n] **of T**, x:T) **ret** i:**nat**
    i:= binary_search_rec(a, x, 1, n)
end fun
{Post: (i $=$ 0 $\Rightarrow$ x no está en a) $\wedge$ (i $\neq$ 0 $\Rightarrow$ x = a[i])}

Búsqueda binaria
Análisis

- Sea $t(n)$ = número de comparaciones que hace en el peor caso cuando el arreglo tiene n celdas.
- $$t(n) = \begin{cases} 0 & \text{si } n = 0 \\ t(n/2) + 1 & \text{si } n > 0 \end{cases}$$
- $a = 1$, $b = 2$ y $k = 0$.
- $a = b^k$.
- conclusión $t(n) \in \mathcal{O}(n^k \log n) = \mathcal{O}(\log n)$.