Divide y vencerás

Idea

- 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: máximo 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) = | 0                          si n ∈ {0,1}
       | t(n/2) + t(n/2) + n-1      si n > 1
```

- 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.

Forma general

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

donde:
- **a**: número de llamadas recursivas a DyV.
- **b**: relación entre el tamaño de x y el de x_i, satisface $|x_i| = |x| / b$
- **k**: el orden de descomponer y combinar el n^k

Conteo

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

```
t(n) = | c                     si la entrada es pequeña o simple
       | a * t(n/b) + g(n)     en caso contrario
```

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 DyV), se llama **recurrencia**. Existen distintos tipos de recurrencia.
Ésta se llama recurrencia **divide y vencerás**.

Recurrencias divide y vencerás

```
Si t(n) = | c                    si la entrada es pequeña o simple
          | a * t(n/b) + g(n)    en caso contrario
```

si t(n) es no decreciente, y g(n) es del orden de n^k, entonces:

$$t(n) \text{ es del orden de } \begin{cases} n^{\log b(a)} & \text{si } a > b^k \\ n^k \log(n) & \text{si } a = b^k \\ n^k \text{ si} & a < b^k \end{cases}$$

Ejemplo: búsqueda binaria

```
{Pre: 1 ≤ lft ≤ n+1 ∧ 0 ≤ rgt ≤ n ∧ a ordenado}
fun binary_search_rec(a: array[1..n] of T, x: T, lft, rgt: nat) ret i: nat
    var mid: nat
    if lft > rgt then
        i:= 0
    else if lft ≤ rgt then
        mid:= (lft+rgt) ÷ 2
        if x < a[mid] then
            i:= binary_search_rec(a, x, lft, mid-1)
        else if x = a[mid] then
            i:= mid
        else if x > a[mid] then
            i:= binary_search_rec(a, x, mid+1,rgt)
        fi
    fi
end fun
{Post: (i = 0 ⇒ x no está en a[lft,rgt]) ∧ (i 6= 0 ⇒ x = a[i])}

{Pre: n ≥ 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 ⇒ x no está en a) ∧ (i 6= 0 ⇒ x = a[i])}
```

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.
- $a = b^k$.
- t(n) es del orden de $n^k \log n$, es decir, del orden de log n.

Funciones según su crecimiento

Análisis de algunos algoritmos

- Ordenación por selección es del orden de n^2.
- Ordenación por inserción es del orden de n^2 (peor caso y caso medio).
- Ordenación por intercalación es del orden de $n \log_2 n$.
- Ordenación rápida es del orden de $n \log_2 n$ (caso medio).
- Búsqueda lineal es del orden de n.
- Búsqueda binaria es del orden de $\log_2 n$.

Cómo comparar los órdenes de los algoritmos?

- Hay funciones que crecen más rápido que otras (cuando n tiende a $+\infty$).
- Escribiremos $f(n) \sqsubset g(n)$ para decir que $g(n)$ crece más rápido que $f(n)$. Por ejemplo:
    - $n \log_2 n \sqsubset n^2$
    - $\log_2 n \sqsubset n$
- Escribiremos $f(n) \approx g(n)$ para decir que $f(n)$ y $g(n)$ crecen al mismo ritmo. Por ejemplo:
    - $(n^2/2) - (n/2) \approx n^2$
    - $45n^2 \approx n^2$

Algunas condiciones

- No nos interesan las constantes multiplicativas:
    - $(1/4)n^2 \approx n^2$
    - $4n^3 \approx n^3$
    - $1000 \log n \approx \log n$
    - $\pi n \approx n$
- No nos interesan los términos menos significativos, que crecen más lento:
    - $n^2 + n \approx n^2$
    - $n^3 + n^2 \log2 n \approx n^3$
    - $\log n + 3456 \approx \log n$
    - $n + \sqrt{n} \approx n$

Cómo comparar funciones según su crecimiento?

Regla del límite. Sean $f(n)$ y $g(n)$ tales que
- $\lim_{n \to +\infty} f(n) = \lim_{n \to +\infty} g(n) = \infty$, y
- $\lim_{n \to +\infty} (f(n)/g(n))$ existe.

Entonces
- si $\lim_{n \to +\infty} (f(n)/g(n)) = 0$, entonces $f(n) \sqsubset g(n)$.
- si $\lim_{n \to +\infty} (f(n)/g(n)) = +\infty$, entonces $g(n) \sqsubset f(n)$.
- caso contrario (el límite es un número real positivo), $f(n) \approx g(n)$.

Jerarquía

Propiedades

- Constantes multiplicativas no afectan.
- Términos de crecimiento despreciable no afectan.
- Sean $a,b > 1$, $\log_a n \approx \log_b n$.
- Sea $f(n) > 0$ para "casi todo $n \in N$". Entonces:
    - $g(n) \sqsubset h(n) \Longleftrightarrow f(n)g(n) \sqsubset f(n)h(n)$.
    - $g(n) \approx h(n) \Longleftrightarrow f(n)g(n) \approx f(n)h(n)$.
- Sea $\lim_{n \to \infty} h(n) = \infty$. Entonces:
    - $f(n) \sqsubset g(n) \Rightarrow f(h(n)) \sqsubset g(h(n))$.
    - $f(n) \approx g(n) \Rightarrow f(h(n)) \approx g(h(n))$.

$$1 \sqsubset \log(\log(\log n)) \sqsubset \log(\log n) \sqsubset \log n \sqsubset n^{0.001} \sqsubset$$
$$\sqsubset n \sqsubset n \log n \sqsubset n^{1.001} \sqsubset n^{100} \sqsubset 1.01^{n} \sqsubset n^{100} * 1.01^{n} \sqsubset$$
$$\sqsubset 1.02^{n} \sqsubset 100^{n} \sqsubset 10000^{n} \sqsubset (n-1)! \sqsubset n! \sqsubset (n+1)! \sqsubset n^{n}$$