Algoritmos y estructuras de datos

www.u-tad.com

Grado en Ingeniería en Desarrollo de Contenidos Digitales

Curso 2015/2016

Introducción

- En este tema vamos a ver las características generales de los algoritmos recursivos…
- … y cómo averiguar su complejidad computacional
  - Matemáticamente habrá que resolver un sistema de ecuaciones

Definición de recursividad I

- Un algoritmo es recursivo cuando el propio algoritmo se llama a sí mismo en algún punto (o varios puntos) de su código
  - Esa llamada se llama “llamada recursiva”
- La recursividad es una forma más natural (que la iteratividad) de ver muchos algoritmos
  - Algunos algoritmos, no obstante, no se pueden hacer recursivos

Definición de recursividad II

- En general:
  - Diremos que un objeto es recursivo si forma parte o se define a si mismo
    - Ej: animaciones o imágenes recursivas

Esquema básico I

- Un algoritmo recursivo siempre sigue un esquema similar:
  - Una condición «si...»
  - Salida del algoritmo en el caso trivial
    - El “caso trivial” es cuando ya no necesitamos hacer una llamada recursiva para resolver el problema
  - En el caso no trivial, se «simplifica» el problema y se llama de nuevo al algoritmo con el problema simplificado como entrada

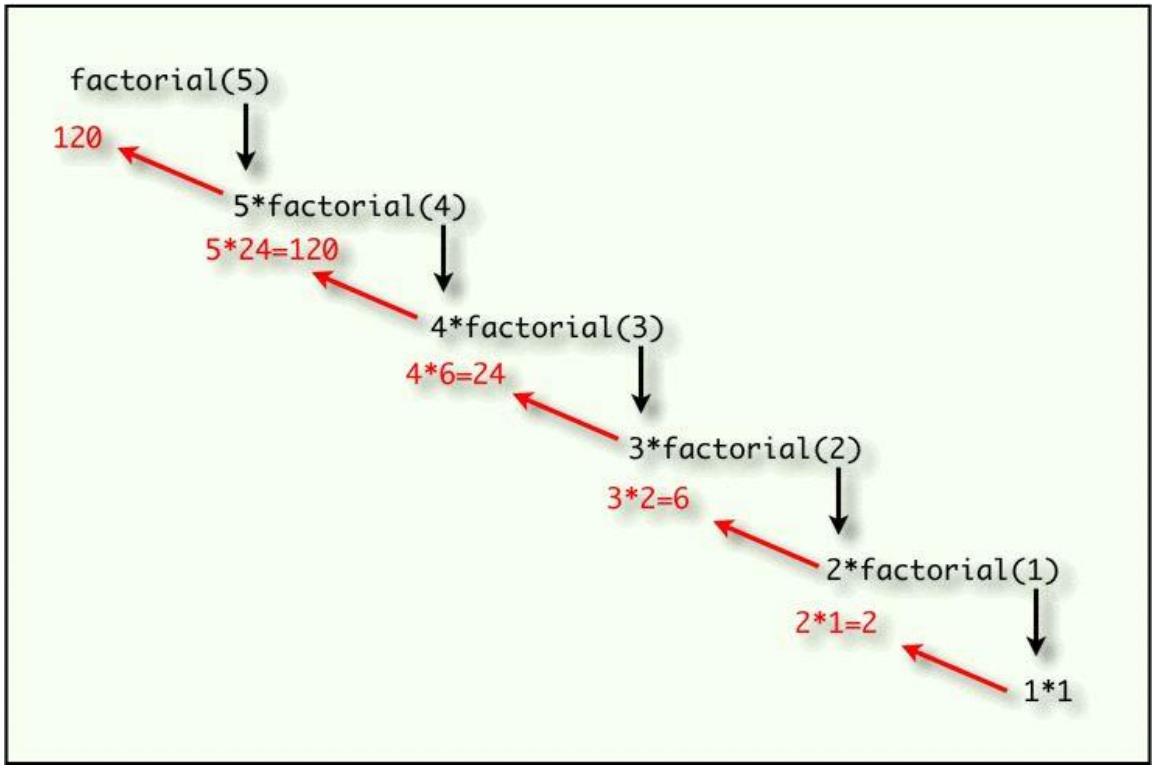
Esquema básico II

- Ejemplo de cálculo recursivo del factorial de un número “n”
  - Recordemos: n! = n·(n-1)·(n-2)·…·1

```
1  algoritmo factorial // recursivo
2  entrada entero n
3  salida entero
4  si n = 0 entonces
5      devolver 1
6  si_no
7      devolver n * factorial ( n − 1 )
```

Esquema básico III

- Para cualquier algoritmo recursivo podemos dibujar un “árbol de llamadas recursivas” para intentar comprender la traza del algoritmo:



Ventajas de la recursividad I

- La recursividad se integra perfectamente y sigue la estrategia de “divide y vencerás” para resolver problemas:
  - Se divide un problema en 2 o más subproblemas (y éstos a su vez en otros subproblemas, y así sucesivamente)
  - Cada subproblema es de naturaleza similar al problema inicial…
  - … pero cada subproblema es de tamaño menor que el problema inicial
  - Se resuelve cada subproblema por separado
  - Una vez resueltos, se combinan sus resultados para producir la solución del problema original

Ventajas de la recursividad II

- A menudo es más fácil realizar un programa recursivo que su equivalente iterativo
  - Porque muchas definiciones y problemas son de naturaleza recursiva…
  - … y, por lo tanto, convertirlos en algoritmo recursivo es inmediato y sencillo

Ventajas de la recursividad III

- Hay ciertas estructuras de datos que se definen recursivamente
  - Listas, árboles, grafos…
  - Al ser recursivas, se manipulan muy fácilmente con algoritmos recursivos
  - Ej: se puede ordenar una lista ordenando cada mitad por separado y luego juntando los resultados (quicksort)

¿Funciona la recursividad? I

- Probar, demostrar o asegurarse de que un algoritmo recursivo funciona es sencillo. Basta con asegurarse de que se cumple todo lo siguiente (**actividad 3.1**):
  - Tiene que existir una salida no recursiva del algoritmo (caso trivial)
    - En el factorial, cuando n es 0
  - Cada llamada recursiva se refiere a un problema más simple
    - La llamada recursiva se hace con n-1, más simple que si la hiciéramos con n
  - Suponiendo que las llamadas recursivas funcionan, el algoritmo debe de funcionar
    - En el factorial, si la llamada recursiva funciona y devuelve (n-1)!, entonces el algoritmo devuelve n!
- Por lo tanto, no hay que hacer ni entender el árbol de llamadas recursivas para entender el algoritmo recursivo en cuestión o para demostrar y/o asegurarse de que efectivamente sí funciona

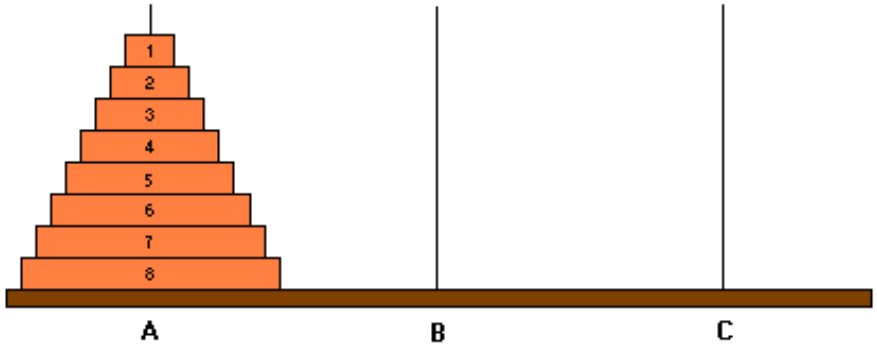
¿Funciona la recursividad? II

- Veamos si hemos comprendido cómo demostrar que un algoritmo recursivo funciona. Usemos para ello el ejemplo de las torres de Hanoi:
  - Tenemos tres postes llamados A, B y C
  - Tenemos n discos, todos ellos de tamaños diferentes
  - Inicialmente tenemos todos los discos en A, de manera que cada disco está encima de todos los que son mayores que él
  - El objetivo es dejar todos los discos en B
  - Reglas:
    - Sólo se puede mover un disco a la vez
    - No se puede poner un disco encima de otro menor
    - Sólo se puede coger el disco más pequeño de cada poste

¿Funciona la recursividad? III

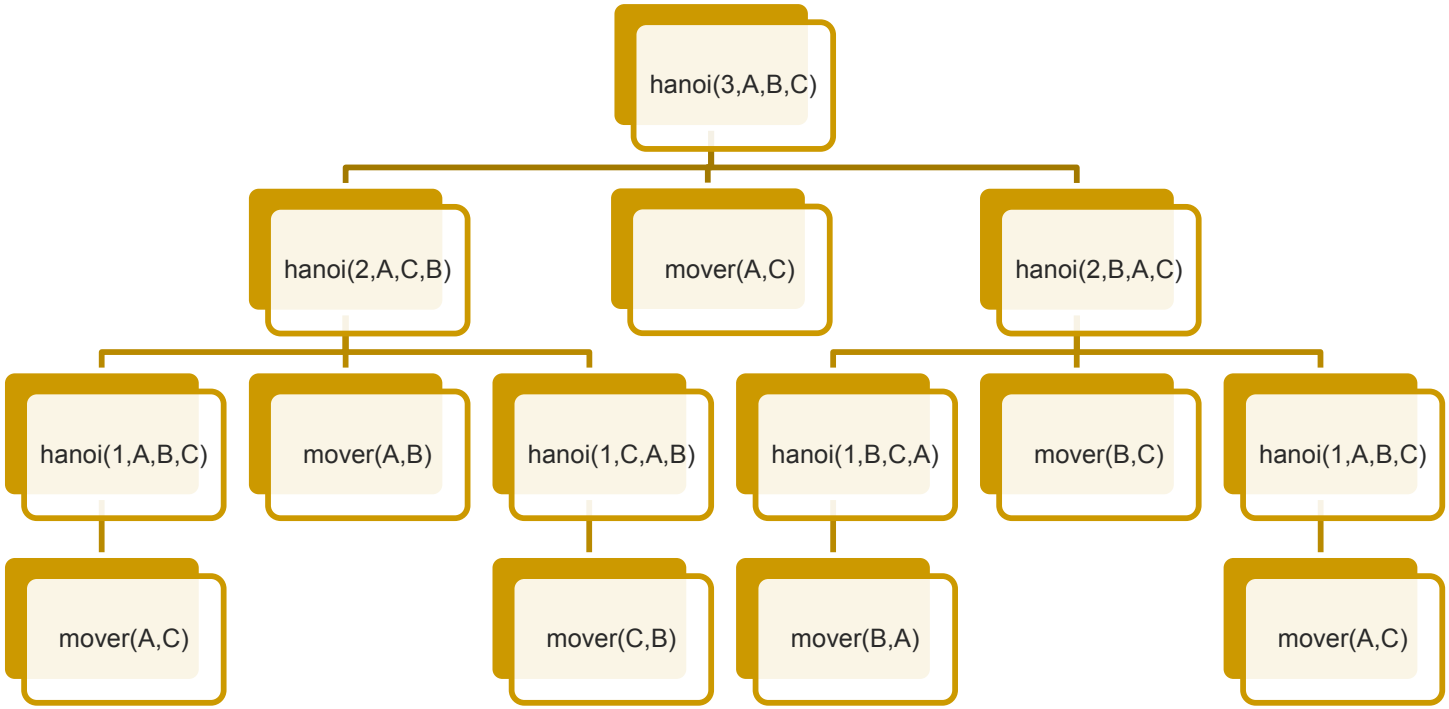
- Pseudocódigo de las Torres de Hanoi (el origen es “A”, el auxiliar es “B” y el destino es “C”):

```
algoritmo hanoi
entrada entero n; poste origen, auxiliar, destino
si n > 0 entonces
    hanoi( n − 1 , origen , destino , auxiliar )
    mover( origen , destino )
    hanoi( n − 1 , auxiliar , origen , destino )
```



¿Funciona la recursividad? IV

- Podemos dibujar el árbol de llamadas recursivas (ej: para n=3), pero no es necesario para comprender que el algoritmo realmente funciona



Programación dinámica I

- Para ilustrar lo que es la programación dinámica, veamos el algoritmo recursivo para calcular el elemento i-ésimo de la sucesión de Fibonacci:

0, 1, 1, 2, 3, 5, 8, 13, 21...

$F(0) = 0, F(1) = 1$

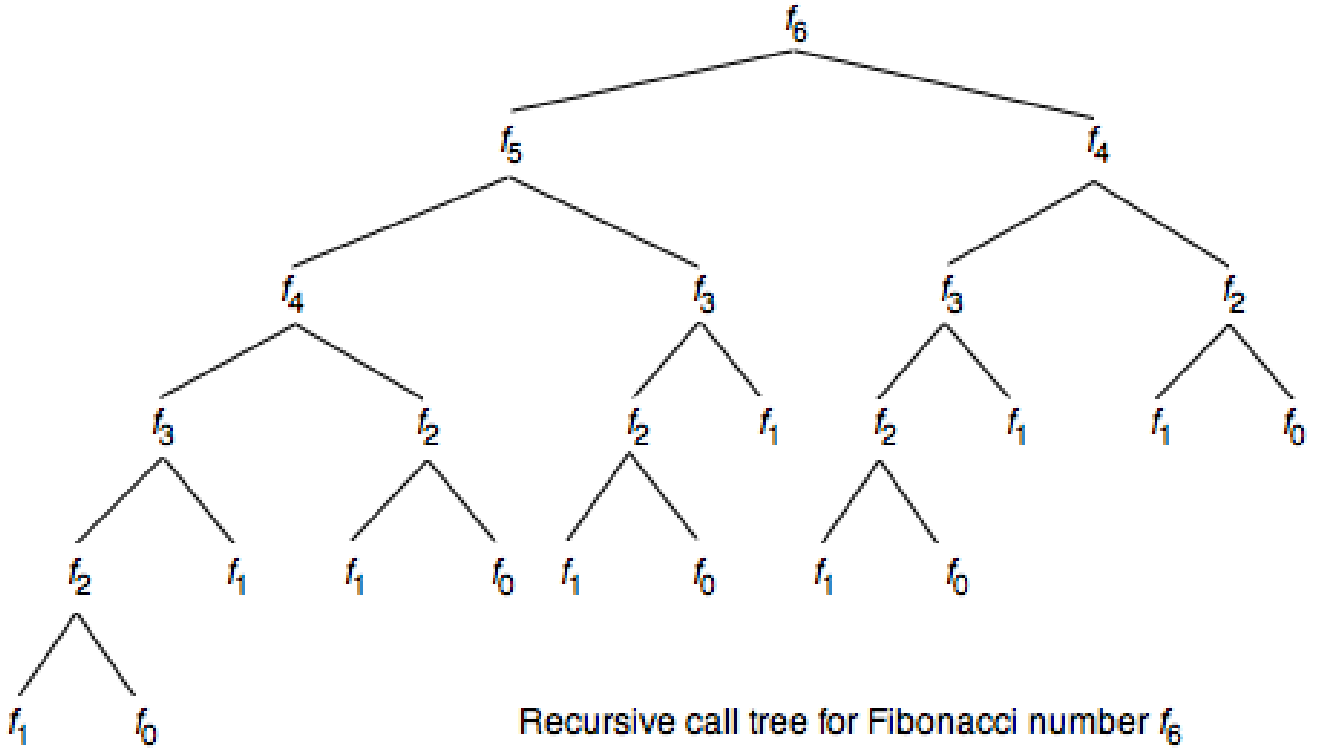
Para $i \geq 2$, $F(i) = F(i-1) + F(i-2)$

Versión recursiva, que surge de manera natural de la definición:

```
algoritmo fibonacci   // recursivo
entrada entero n
salida  entero
si  n < 2 entonces
    devolver n
si_no
    devolver fib( n − 1 ) + fib( n − 2 )
```

Programación dinámica II

- Árbol de llamadas recursivas del cálculo recursivo del séptimo elemento de la serie de Fibonacci:



Recursive call tree for Fibonacci number f_6

Programación dinámica III

- Fibonacci (cont.):
  - En el árbol de llamadas recursivas vemos que los mismos valores se calculan multitud de veces de forma redundante
    - Por lo tanto la programación recursiva de Fibonacci es el ejemplo más típico de cuando no utilizar la recursión, ya que se calcula varias veces el mismo valor
    - Se soluciona con la “programación dinámica”: una vez calculada una solución, guardarla para su posible uso posterior
      - Evitar calcular varias veces el mismo resultado, manteniendo una tabla de resultados conocidos

T(n) y M(n) de algoritmos recursivos I

- El T(n) o M(n) de un algoritmo recursivo es una función/ecuación en recurrencia
  - Una función en recurrencia o recurrente es aquélla que se incluye a sí misma en su definición
  - Todas las funciones recurrentes tienen:
    - Una o varias condiciones iniciales
      - Las sacaremos del caso trivial
    - Una expresión general
      - La sacaremos del caso no trivial
  - Veamos algunos ejemplos

T(n) y M(n) de algoritmos recursivos II

- ¿Cuánto tarda en ejecutarse el algoritmo recursivo para calcular el factorial de un número “n”?
  - T(0)=1 (condición inicial sacada del caso trivial)
  - T(n)=1+T(n-1) para n>=1 (condición general sacada del caso no trivial)

```
1  algoritmo factorial   // recursivo
2  entrada entero n
3  salida entero
4  si n = 0 entonces
5     devolver 1
6  si_no
7     devolver n * factorial ( n − 1 )
```

T(n) y M(n) de algoritmos recursivos III

- ¿Cuánta memoria adicional usa el algoritmo recursivo para calcular el factorial de un número “n”?
  - M(0)=1 (condición inicial sacada del caso trivial)
  - M(n)=1+M(n-1) para n>=1 (condición general sacada del caso no trivial)
  - Por casualidad, en este algoritmo M(n) coincide con T(n)
    - Pero no tiene por qué ser siempre así
  - Recordemos que cualquier función siempre usa algo de memoria adicional, al menos para recordar a dónde tiene que saltar en cuanto termine
    - Por lo tanto, siempre al menos será M(n)=1

```
1  algoritmo factorial // recursivo
2  entrada entero n
3  salida entero
4  si n = 0 entonces
5      devolver 1
6  si_no
7      devolver n * factorial ( n - 1 )
```

T(n) y M(n) de algoritmos recursivos IV

- Algoritmo recursivo para resolver Hanoi:
  - Función de tiempo:
    - T(0)=1
    - T(n)=2·T(n-1)+1 para n>=1
      - Suponemos que mover un disco de un poste a otro es una operación elemental
  - Función de memoria adicional:
    - M(0)=1
    - M(n)=2·M(n-1)+1 para n>=1

```
algoritmo hanoi
entrada entero n; poste origen, auxiliar, destino
si n > 0 entonces
    hanoi( n − 1 , origen , destino , auxiliar )
    mover( origen , destino )
    hanoi( n − 1 , auxiliar , origen , destino )
```

T(n) y M(n) de algoritmos recursivos V

- Algoritmo recursivo para calcular el componente n-ésimo de la serie de Fibonacci:
  - Función de tiempo:
    - T(0)=1; T(1)=1
    - T(n)=T(n-1)+T(n-2)+1 para n>=2
  - Función de memoria adicional:
    - M(0)=1; M(1)=1
    - M(n)=M(n-1)+M(n-2)+1 para n>=2

```
algoritmo fibonacci   // recursivo
entrada entero n
salida  entero
si  n < 2 entonces
    devolver n
si_no
    devolver fib( n − 1 ) + fib( n − 2 )
```

Eliminar recurrencia sencilla I

- De una función en recurrencia no podemos obtener directamente su complejidad, por lo cual tenemos que convertirla en una función no recurrente
  - Es decir, una función que no dependa de sí misma
  - A partir de la función no recurrente equivalente ya podríamos obtener su O(…) según las reglas vistas

Eliminar recurrencia sencilla II

- Para eliminar la recurrencia en una ecuación recurrente, hay varios casos
  - Vamos a ver primero el más sencillo, en el cual la función recurrente se puede ajustar al siguiente aspecto:
    - $0=a_0F(n)+a_1F(n-1)+\ldots+a_kF(n-k)$
    - Esta forma se llama “ecuación homogénea”

Eliminar recurrencia sencilla III

Dada una ecuación en recurrencia con el siguiente aspecto:

$$a_0 F_n + a_1 F_{n-1} + \ldots + a_k F_{n-k} = 0$$

que denominaremos ecuación homogénea

Buscaremos soluciones de la forma $F_n = r^n$ donde r es una constante

Entonces:

$$a_0 r^n + a_1 r^{n-1} + \ldots + a_k r^{n-k} = 0$$

$$r^{n-k}(a_0 r^k + a_1 r^{k-1} + \ldots + a_k) = 0$$

Eliminar recurrencia sencilla IV

Dejando aparte la solución trivial de $r = 0$:

$$a_0 r^k + a_1 r^{k-1} + \ldots + a_k = 0$$

que denominaremos ecuación característica.

Si ese polinomio tiene k raíces distintas, entonces

$$F_n = c_1 r_1^n + c_2 r_2^n + \ldots + c_k r_k^n$$

es una solución de la ecuación homogénea.

Los c_i se encontrarán a partir de las condiciones iniciales.

Eliminar recurrencia sencilla VI

Ejemplo *Para Fibonacci teníamos* $F_n - F_{n-1} - F_{n-2} = 0$. *La ecuación característica es* $r^2 - r - 1 = 0$, *por lo tanto* $r_1 = \frac{1+\sqrt{5}}{2}$ *y* $r_2 = \frac{1-\sqrt{5}}{2}$. *Así, las soluciones de la ecuación homogénea serán de la forma:*

$$F_n = c_1 \left(\frac{1+\sqrt{5}}{2}\right)^n + c_2 \left(\frac{1-\sqrt{5}}{2}\right)^n$$

Sabiendo que $F_0 = 0$ *y que* $F_1 = 1$ *obtenemos que* $c_1 = \frac{1}{\sqrt{5}}$ *y que* $c_2 = -\frac{1}{\sqrt{5}}$. *Entonces*

$$F_n = \frac{1}{\sqrt{5}} \left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n \right)$$

Eliminar recurrencia sencilla VII

Si no todas las raíces son distintas, y r_i es una raíz con multiplicidad m, entonces $r_i^n, nr_i^n, n^2r_i^n, \ldots n^{m-1}r_i^n$ formarán parte de la base.

Ejemplo

$$F_0 = 0, F_1 = 1, F_2 = 2$$

$$F_n = 5F_{n-1} - 8F_{n-2} + 4F_{n-3} \text{ si } n \geq 3$$

La ecuación característica es $r^3 - 5r^2 + 8r - 4 = 0$. *Entonces* $(r-1)(r-2)^2 = 0$. *Por lo tanto:*

$$F_n = c_1 1^n + c_2 2^n + c_3 n 2^n$$

y utilizando las condiciones iniciales

$$F_n = -2 + 2^{n+1} - n2^{n-1}$$