Recursion

# Recursión

A *recursive* función es una función que se llama a sí misma, directa o indirectamente. La recursión es una potente técnica de programación en la que un problema se divide en un conjunto de subproblemas similares, cada uno de los cuales se resuelve con una solución trivial. Generalmente, una función recursiva se llama a sí misma para resolver sus subproblemas.

Las Torres de Hanoi es un famoso rompecabezas. El equipo incluye tres postes y un conjunto de discos de varios diámetros con agujeros en sus centros. El equipo apila todos los discos en el poste de origen, con los discos más pequeños encima de los más grandes. El objetivo es desplazar la pila hasta el poste de destino moviendo un disco cada vez a otro poste, sin colocar nunca un disco más grande sobre un disco más pequeño. Este puzzle tiene una solución recursiva trivial:

var hanoi = function hanoi(disc, src, aux, dst) {  
 if (disc > 0) {  
 hanoi(disc − 1, src, dst, aux);  
 document.writeln('Move disc ' + disc +  
 ' from ' + src + ' to ' + dst);  
 hanoi(disc − 1, aux, src, dst);  
 }  
};  
  
hanoi(3, 'Src', 'Aux', 'Dst');

Produce esta solución para tres discos:

Move disc 1 from Src to Dst  
Move disc 2 from Src to Aux  
Move disc 1 from Dst to Aux  
Move disc 3 from Src to Dst  
Move disc 1 from Aux to Src  
Move disc 2 from Aux to Dst  
Move disc 1 from Src to Dst

La función hanoi mueve una pila de discos de un poste a otro, utilizando el poste auxiliar si es necesario. Divide el problema en tres subproblemas. En primer lugar, descubre el disco inferior moviendo la subpila que tiene encima hasta el poste auxiliar. A continuación, puede desplazar el disco inferior al poste de destino. Por último, puede mover la subpila desde el poste auxiliar hasta el poste de destino. El movimiento de la subpila se gestiona llamándose a sí misma recursivamente para resolver esos subproblemas.

A la función hanoi se le pasa el número del disco que debe mover y los tres puestos que debe utilizar. Cuando se llama a sí misma, debe tratar el disco que está por encima del disco en el que está trabajando en ese momento. Eventualmente, se le llamará con un número de disco inexistente. En ese caso, no hace nada. Ese acto de no hacer nada nos da la seguridad de que la función no recurre eternamente.

Las funciones recursivas pueden ser muy eficaces en la manipulación de estructuras de árbol como el Modelo de Objetos del Documento (DOM) del navegador. A cada llamada recursiva se le da un trozo más pequeño del árbol sobre el que trabajar:

// Define a walk\_the\_DOM function that visits every  
// node of the tree in HTML source order, starting  
// from some given node. It invokes a function,  
// passing it each node in turn. walk\_the\_DOM calls  
// itself to process each of the child nodes.  
  
var walk\_the\_DOM = function walk(node, func) {  
 func(node);  
 node = node.firstChild;  
 while (node) {  
 walk(node, func);  
 node = node.nextSibling;  
 }  
};  
  
// Define a getElementsByAttribute function. It  
// takes an attribute name string and an optional  
// matching value. It calls walk\_the\_DOM, passing it a  
// function that looks for an attribute name in the  
// node. The matching nodes are accumulated in a  
// results array.  
  
var getElementsByAttribute = function (att, value) {  
 var results = [];  
  
 walk\_the\_DOM(document.body, function (node) {  
 var actual = node.nodeType === 1 && node.getAttribute(att);  
 if (typeof actual === 'string' &&  
 (actual === value || typeof value !== 'string')) {  
 results.push(node);  
 }  
 });  
  
 return results;  
};

Algunos lenguajes ofrecen la función *tail recursion optimization*. Esto significa que si una función devuelve el resultado de invocarse a sí misma recursivamente, entonces la invocación se sustituye por un bucle, lo que puede acelerar significativamente las cosas. Lamentablemente, JavaScript no proporciona actualmente una optimización de la recursividad de cola. Las funciones que recurren muy profundamente pueden fallar al agotar la pila de retorno:

// Make a factorial function with tail  
// recursion. It is tail recursive because  
// it returns the result of calling itself.  
  
// JavaScript does not currently optimize this form.  
  
var factorial = function factorial(i, a) {  
 a = a || 1;  
 if (i < 2) {  
 return a;  
 }  
 return factorial(i − 1, a \* i);  
};  
  
document.writeln(factorial(4)); // 24