

**ESCUELA SUPERIOR POLITECNICA DE CHIMBORAZO**

**FACULTAD DE INFORMATICA Y ELECTRÓNICA**

**ESCUELA DE INGENIERÍA EN SISTEMAS**

**Nombre: Wilmer Aldas-Claudio Pilataxi**

**Curso: 6° “A”**

**Código:6035**

**Fecha:12/12/16**

En muchos casos, la recursividad requiere más recursos (tiempo y memoria) para resolver el problema que una versión iterativa, por lo que, independientemente de la naturaleza del problema, la recursividad tiene su gran aplicación para problemas complejos cuya solución recursiva es más fácil de obtener, más estructurada y sencilla de mantener. Así por ejemplo, los problemas de calculó de factorial o de búsqueda binaria son fáciles de programar iterativamente por lo generalmente no se utilizan las versiones recursivas:

• En el caso del factorial, no necesitamos más que un bucle para calcular el producto mientras que en la versión recursiva el número de llamadas que se anidarían es igual que el valor de entrada de manera que usar la versión recursiva es más un error que algo poco recomendable.

• Para la búsqueda binaria el máximo número de llamadas anidadas (profundidad de recursión) es del orden del logaritmo del tamaño del vector por lo que este factor es menos importante. A pesar de ello, y considerando que la versión iterativa es muy simple, también para este caso podemos rechazar el uso de la recursividad.

Incluso para casos más complejos pero correspondientes a funciones que se llaman muchas veces en un programa puede ser interesante eliminar la recursividad, ya sea de una manera sencilla como es el caso de la recursión de cola o de forma más compleja mediante la introducción de estructuras adicionales como pilas definidas por el programador.

Además, cuando se resuelven problemas de manera recursiva hay que tener especial cuidado ya que el que la función tenga varias llamadas recursivas puede hacer que el número de subproblemas a resolver crezca de forma exponencial haciendo incluso que el resultado obtenido sea mucho más ineficiente. Por tanto, podemos plantear una ´ultima regla en la programación recursiva: No resolver varias veces el mismo subproblema en llamadas recursivas distintas.

A pesar de estas consideraciones, existen multitud de problemas que se ven radicalmente simplificados mediante el uso de la recursividad y que cuya diferencia en recursos necesarios con respecto a la solución iterativa puede ser obviada a efectos prácticos.

Memoization

La estrategia más intuitiva para recordar resultados calculados es definir una estructura de memoria externa a la función, y aprovecharla dentro de la función para recordar resultados. Esta estrategia se llama memoización ("memoization" en inglés, no a confundir con "memorization"). En el caso de la serie Fibonacci el código sería el siguiente:

vector<int> res(100, -1); // inicializar el resultado a -1

res[0] = res[1] = 1; // establecer los casos base

int fibonacci(int n) {

if (res[n] > 0) return res[n];

else return res[n] = fibonacci(n - 1) + fibonacci(n - 2);

}

La primera vez que se llama a Fibonacci con n=2, el valor de res[2] es -1 (el valor inicial). Por lo tanto, no se cumple la condición del if, por lo que recursivamente se llama a fibonacci con n=1 y n=0, respectivamente. Para n=1 y n=0, el valor de res[n] es 1, que es mayor que 0, por lo que directamente se devuelve. La suma fibonacci(1)+fibonacci(0) = 1+1 = 2 se guarda en res[2] y se devuelve.

La siguiente vez que se llama a Fibonacci con n=2, el valor de res[2] es 2, por lo que se cumple la condición del if. Como consecuencia se devuelve 2 directamente sin llamar recursivamente a Fibonacci con n=1 y n=0.

Con esta modificación la complejidad del algoritmo disminuye de exponencial a lineal, lo que significa un ahorro enorme de tiempo. (Para la serie Fibonacci existe un algoritmo que calcula el n-ésimo número en tiempo logarítmico, que es mucho más rápido todavía.)

Los números de Fibonacci aparecen en muchos temas relacionados con la enseñanza de la informática como la eliminación de la recursividad y la memoizacion.

Bibliografía

Algoritmos recursivos

<https://books.google.com.ec/books?id=OC17arE5xukC&pg=PA307&lpg=PA307&dq=como+eliminar+la+recursividad+en+programacion&source=bl&ots=pupNqpDrEy&sig=OsCIN3nAzg7fA7INhHVEmXNStrA&hl=es&sa=X&ved=0ahUKEwir7KD68-_QAhUF6yYKHf1NBoYQ6AEINzAF#v=onepage&q=como%20eliminar%20la%20recursividad%20en%20programacion&f=false>

memoization

<http://www.dtic.upf.edu/~jonsson/acm/dp/>