**ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO**

**FACULTAD DE INFORMATICA Y ELECTRONICA**

**ESCUELA DE INGENIERIA EN SISTEMAS**

**PROGRAMACIÓN AVANZADA**

**INTEGRANTES:** Marcelo Romero (6030)

Jorge Benítez (6039)

Lizbeth Cando (5951)

Fabián Banderas (5759)

**TEMA:** Estrategias de memorización y tabulación empleadas para eliminar la recursividad de un algoritmo.

Se dice que algo es recursivo si se define en función de sí mismo o a sí mismo. También se dice que nunca se debe incluir la misma palabra en la definición de ésta.

Un requisito importante para que sea correcto un algoritmo recursivo es que no genere una secuencia infinita de llamadas así mismo. Claro que cualquier algoritmo que genere tal secuencia no termina nunca. Una función recursiva f debe definirse en términos que no impliquen a f al menos en un argumento o grupo de argumentos. Debe existir una "salida" de la secuencia de llamadas recursivas.

Si en esta salida no puede calcularse ninguna función recursiva. Cualquier caso de definición recursiva o invocación de un algoritmo recursivo tiene que reducirse a la larga a alguna manipulación de uno o casos más simples no recursivos.

**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 memorización ("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.)

**Linkcografia**

[**file:///C:/Users/USER/Downloads/recursividad.pdf**](file:///C:/Users/USER/Downloads/recursividad.pdf)

[**http://stackoverflow.com/questions/7875380/recursive-fibonacci-memoization**](http://stackoverflow.com/questions/7875380/recursive-fibonacci-memoization)

[**https://www.google.com.ec/url?sa=t&rct=j&q=&esrc=s&source=web&cd=5&cad=rja&uact=8&ved=0ahUKEwjh8YG4pPDQAhVGPiYKHa3JAoUQFgg1MAQ&url=http%3A%2F%2Ffiles.unefa-io.webnode.es%2F200000021-499594a8f4%2FUnidad7.pdf&usg=AFQjCNGklbjt9Gih3h8QzAJiYniUfhhqjA&bvm=bv.141320020,d.eWE**](https://www.google.com.ec/url?sa=t&rct=j&q=&esrc=s&source=web&cd=5&cad=rja&uact=8&ved=0ahUKEwjh8YG4pPDQAhVGPiYKHa3JAoUQFgg1MAQ&url=http%3A%2F%2Ffiles.unefa-io.webnode.es%2F200000021-499594a8f4%2FUnidad7.pdf&usg=AFQjCNGklbjt9Gih3h8QzAJiYniUfhhqjA&bvm=bv.141320020,d.eWE)