Suarez Vázquez Brandon Abraham 17211566

Recursividad

Definición

Recursividad se puede definir como una técnica que permite que una función se llame así misma.

Son Recursivos aquellos algoritmos que, estando encapsulados dentro de una función, son llamados desde ella misma una y otra vez, en contraposición a los algoritmos que hacen uso de bucles while, do-while, for.

Tipos.

Podemos distinguir dos tipos de recursividad:

Recursividad Múltiple: Se da cuando hay más de una llamada a sí misma dentro del cuerpo de la función, resultando más difícil de hacer de forma iterativa.

Recursividad Simple: Aquella en cuya definición sólo aparece una llamada recursiva.

Se puede transformar con facilidad en algoritmos iterativos.

Características.

Consta de una parte recursiva, otra iterativa o no recursiva y una condición de terminación. La parte recursiva y la condición de terminación siempre existen. En cambio, la parte no recursiva puede coincidir con la condición de terminación.

Algo muy importante a tener en cuenta cuando usemos la recursividad es que es necesario asegurarnos que llega un momento en que no hacemos más llamadas recursivas. Si no se cumple esta condición el programa no parará nunca.

Ventaja.

La principal ventaja es la simplicidad de comprensión y su gran potencia, favoreciendo la resolución de problemas de manera natural, sencilla y elegante; y facilidad para comprobar y convencerse de que la solución del problema es correcta.

Inconveniente.

El principal inconveniente es la ineficiencia tanto en tiempo como en memoria, dado que para permitir su uso es necesario transformar el programa recursivo en otro iterativo, que utiliza bucles y pilas para almacenar las variables.

Aplicaciones

La recursividad es una herramienta muy potente en algunas aplicaciones, sobre todo de cálculo.

Para estimar el tiempo de ejecución: muchas veces el tiempo de ejecución de un algoritmo se expresa de manera natural como una función recursiva. Especialmente los algoritmos de tipo "divide y vencerás" donde tratas de resolver una instancia de tamaño n dividiéndola en k partes de aproximadamente el mismo tamaño n/b y luego juntando las soluciones con un método que tarda el orden de n^d pasos:

$T(n) = k\,T(n/b) + O(n^d)$

Para definir complicadas estructuras de datos: Muchas veces quieres definir nuevos tipos de dato para manipular grandes cantidades de información. Ejemplo de esto son los "montículos de Fibonacci" que organizan la información de tal manera que siempre tienes acceso al elemento más pequeño de forma inmediata. Su definición es muy laboriosa, pero se entiende fácilmente luego de que estudias la definición recursiva de un árbol de Fibonacci.

Para analizar las limitaciones teóricas: Esto es muy útil en la práctica porque si ya sabes que tu problema es indecidible, entonces ya no pierdes más tiempo buscando un algoritmo para solucionarlo; en lugar de eso seguramente tratarás de simplificar el problema.

Para definir nuevos lenguajes de programación: La sintaxis de un lenguaje de programación se define (salvo muy pocas excepciones) de manera recursiva. Por ejemplo, si quisieras definir una oración del español pondrías algo como ORACIÓN = SUJETO + VERBO + PREDICADO y luego defines recursivamente cada una de las partes.

La razón de que existan lenguajes que admitan la recursividad se debe a la existencia de estructuras específicas tipo pilas (stack) para este tipo de procesos y memorias dinámicas.

La recursividad es una manera elegante, intuitiva y concisa de plantear una solución, y es uno de los pilares de la programación funcional. Pero no quiere decir que sea un sistema eficiente, es decir, que el número de operaciones que hay que hacer sea inferior que si se utiliza otra forma de resolverlo, como sería mediante una estructura repetitiva. Al contrario, la recursividad es cara y exige un mayor procesamiento.

Fuentes:

http://www.abacusnt.es/tem26.pdf

http://www.monografias.com/trabajos10/esda/esda.shtml#recu

http://ar.answers.yahoo.com/question/index?qid=20110418160623AAba8rt