

Capítulo 4

Recursividad.

4.1. Competencias

Las competencias a desarrollar en el tema son:

- **CTEC3** Capacidad para evaluar la complejidad computacional de un problema, conocer estrategias algorítmicas que puedan conducir a su resolución y recomendar, desarrollar e implementar aquella que garantice el mejor rendimiento de acuerdo con los requisitos establecidos.

4.2. Introducción.

En la asignaturas previas de Programación y de Estructuras de Datos, la recursividad se ha analizado desde un punto de vista eminentemente práctico, pero no se han estudiado con profundidad las ventajas e inconvenientes de la misma. En este capítulo se estudiará la recursividad de una forma más rigurosa y exhaustiva, analizando sus ventajas e inconvenientes y desarrollando una serie de ejemplos ilustrativos.

La recursividad o recursión es una técnica de programación que permite simplificar ciertos programas que se basan en la repetición anidada de ciertos procedimientos con diferentes parámetros. La idea consiste en la resolución de un problema para un valor concreto de sus datos a partir de la solución del problema para otros datos más simples, que en la mayoría de las ocasiones representan un subconjunto de los datos iniciales o un valor más pequeño del dato o datos iniciales (caso del factorial recursivo). Por tanto, será aplicable a aquellos problemas que puedan ser reducidos a problemas similares pero para un subconjunto de los datos iniciales o para datos más pequeños. Normalmente, programando dicha reducción se resolverá el problema sin necesidad de especificar la solución de los subproblemas en los que se descompone el problema original, pues esta labor ya se ha realizado al ser tales

subproblemas casos particulares del problema resuelto. Por ejemplo, en el cálculo del factorial, la reducción se podría expresar como:

$$factorial(n) = n * factorial(n - 1) \quad (4.1)$$

y el caso particular ya resuelto sería:

$$factorial(1) = 1 \quad (4.2)$$

Se puede concluir que la clave del diseño de un algoritmo recursivo es la descomposición del problema en subproblemas similares sobre datos más simples que los del problema original. Al descomponer los problemas sucesivamente en subproblemas cada vez más simples, se llega a subproblemas tan sencillos que sus soluciones son obvias, por lo que se puede dar la solución a los mismos directamente, sin aplicar ningún algoritmo.

4.3. Ventajas e inconvenientes de la recursividad.

Para analizar los inconvenientes de la recursividad, es necesario observar de qué forma se ejecuta en un ordenador un procedimiento recursivo. El ordenador realiza cálculos secuenciales basados en la iteración, simulando las llamadas recursivas haciendo uso de la pila, lo cual implica un uso ineficiente de la memoria en comparación con un algoritmo iterativo, ya que en cada llamada recursiva se crean nuevas variables locales que requieren un mayor uso de la memoria dinámica, cosa que no ocurre en un procedimiento iterativo, en el cual las variables del entorno son las mismas en cada una de las iteraciones. De esta forma, el ordenador traduce un procedimiento recursivo a iterativo de forma automática. Evidentemente, esta *traducción* la podría realizar el programador, codificando de forma directa un procedimiento iterativo, ahorrándole tiempo de cómputo al ordenador al evitar esa *traducción*, y obteniendo una versión más rápida y eficiente en cuanto al uso de la memoria. Este aspecto podría considerarse un inconveniente de la recursividad, aunque este inconveniente puede ser muy relativo, ya que aunque elaboremos una versión iterativa y ésta funcione, no hay que olvidar que en muchas ocasiones se ha usado la recursividad para desarrollar el procedimiento original. De esta forma, es cierto que se ahorra tiempo de cómputo al ordenador pero a costa del tiempo empleado por el programador en esa *traducción*. Por dicha razón, la desventaja citada no se puede considerar como una desventaja concluyente.

En realidad, los inconvenientes de la recursión se derivan de la forma en que el ordenador trata a los procedimientos recursivos. Dado que el ordenador convierte las llamadas recursivas en cálculos secuenciales, los subproblemas que se van obteniendo son tratados localmente, cada uno por separado y sin tener en cuenta si ese

procedimiento ya se ha ejecutado antes. Por esta razón, en aquellos problemas en los que el planteamiento recursivo genera llamadas recursivas idénticas, el ordenador repite estas llamadas sin tener en cuenta que se están repitiendo una y otra vez, utilizando tiempos de cómputo que en muchas ocasiones no son abordables, además de que la pila crece de la misma forma que las llamadas recursivas, creando problemas por el posible desborde de la memoria dinámica. Esto sucede, por ejemplo, en el algoritmo recursivo que obtiene el término n -ésimo de la sucesión de Fibonacci.

Algoritmo *Fibonacci*(n)

inicio

si $n \leq 2$ **entonces**

devolver $n - 1$

sino

devolver *Fibonacci*($n - 1$) + *Fibonacci*($n - 2$)

finsi

fin

Basta con analizar superficialmente las llamadas recursivas, para advertir que se producen dos llamadas con el valor $n - 2$, tres llamadas con el valor $n - 3$, y a medida que disminuye el valor de n , el número de llamadas recursivas repetidas crece exponencialmente. Si una persona tuviese que simular a mano un procedimiento recursivo con tales características, advertiría estas llamadas repetidas, y las ejecutaría una sola vez guardando el resultado de las mismas, y en cuanto que se produjese una llamada repetida, ésta llamada sería sustituida por el valor previamente calculado y almacenado. Por contra, el ordenador ignoraría tales repeticiones de una misma llamada. El número de repeticiones innecesarias se puede desbordar, dando lugar a tiempos de cómputo elevadísimos. Por ejemplo, en la sucesión de Fibonacci, el número de llamadas será del orden de 2^n siendo n el orden del término que se quiere calcular. Esto implica que para un n no muy grande, del orden de 50 por ejemplo, el tiempo de cómputo se eleva considerablemente. Estas repeticiones se podrían evitar implementando la forma de actuar del humano, es decir, almacenando en una estructura de datos adicional (una tabla o una lista) el resultado de las llamadas que ya han sido ejecutadas y sus efectos. De esta forma, antes de ejecutar una llamada, se comprueba si ésta ya ha sido ejecutada, y en caso afirmativo se obtendrá el elemento correspondiente de la estructura de datos adicional y éste sustituirá a la llamada. Esta solución puede tener el inconveniente de que si se generan llamadas para un rango muy amplio de valores, la estructura donde se almacenen las llamadas y sus efectos podría tener muchos elementos lo cual la hará más ineficiente debido a los elevados tiempos de búsqueda en la misma y a la alta capacidad de almacenamiento requerida.

Para finalizar, podemos destacar las siguientes conclusiones:

- Por lo general, el uso de la recursividad es menos conveniente, tanto desde el punto de vista del uso de memoria como desde el del tiempo de ejecución.

- En la mayoría de los problemas que admiten una solución recursiva, dicha solución es mucho más evidente y fácil de obtener. Por otra parte, hay que destacar que si la solución recursiva implica más de una llamada recursiva, la solución recursiva es aún mucho más simple que la iterativa.
- La simplicidad e inmediatez también son características deseables para un algoritmo, ya que el hecho de pasar un algoritmo recursivo a iterativo implica un consumo de tiempo, y además esta *traducción* no siempre es fácil, sobre todo cuando hay dos o más llamadas recursivas, como se ha resaltado en el punto anterior.
- Si tenemos un algoritmo recursivo sólo debemos traducirlo a forma iterativa en caso de que la versión iterativa sea mucho más eficiente, y además dicha versión no sea muy complicada de implementar. Éste podría ser el caso en el que el número de llamadas recursivas repetidas sea inabordable y no se puedan evitar, como es el caso del determinante recursivo, ya citado en el tema 1 como ejemplo de algoritmo ineficiente.

4.4. Ejemplos

4.4.1. El juego de la rayuela

Es uno de los ejemplos típicos donde el planteamiento recursivo es evidente y fácil de implementar. El juego de la rayuela consiste en lo siguiente:

- Se tiene una hilera de adoquines numerados secuencialmente a partir de cero.
- Se permiten dos tipos de movimientos para desplazarse por los adoquines:
 - Ir al adoquín siguiente.
 - Ir al adoquín que hay detrás del siguiente.
- El juego comienza en el adoquín cero.
- El juego termina cuando se llega al último adoquín.

El problema consiste en determinar cuántas caminos posibles existen para ir del adoquín 0 al n . La solución a este problema se puede plantear de forma recursiva. Para ello planteamos el caso base o elemental, que ya está resuelto, y el caso general que es el que contiene la descomposición recursiva. Los casos particulares se obtienen sabiendo que solo hay una forma posible de llegar al adoquín 1, y dos formas posibles de llegar al adoquín 2. El caso general se formula sabiendo que, en función de la formulación del problema, al adoquín n solo se puede acceder de

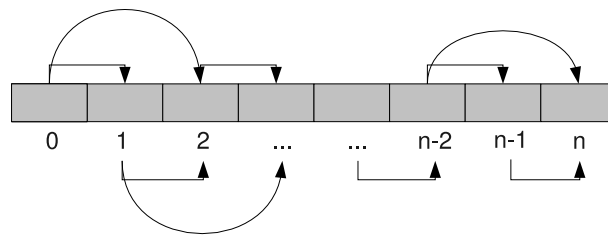


Figura 4.1: Juego de la rayuela

forma directa desde el adoquín $n - 1$ y del $n - 2$, por tanto el número de caminos para acceder al adoquín n es la suma de los caminos para acceder al $n - 1$ y al $n - 2$

- Caso elemental de la recursión:

$$\text{numeroCaminos}(1) = 1$$

$$\text{numeroCaminos}(2) = 2$$

- Caso general:

$$\text{numeroCaminos}(n) = \text{numeroCaminos}(n-1) + \text{numeroCaminos}(n-2)$$

El algoritmo sería similar al de Fibonacci, con lo cual tendría el mismo inconveniente de la repetición de las llamadas recursivas.

Algoritmo *Rayuela*(n)

inicio

si $n \leq 2$

devolver n

sino

devolver $\text{Rayuela}(n - 1) + \text{Rayuela}(n - 2)$

finsi

fin

4.4.2. El plano de la ciudad.

Para enunciar el problema, primero supondremos que en una ciudad la disposición de las calles es de forma tal que una calle es siempre paralela o perpendicular a otra calle, y el plano lo vamos a representar en el primer cuadrante de un sistema de coordenadas plano, de forma tal que forman una malla rectangular en la cual los puntos de la malla son las intersecciones de las calles. Supongamos por otra parte, que un automovilista desea ir desde el origen de coordenadas $O(0, 0)$ a un punto de

la malla $P(x, y)$ de la forma más rápida posible. Es evidente que su estrategia se ha de basar en dos movimientos elementales:

- Desplazarse al punto inmediatamente a la derecha del punto en que se encuentra.
- Desplazarse al punto inmediatamente superior al que se encuentra.

Esta estrategia asegura que no se saldrá del plano y que llegará de la forma más rápida posible. El problema consiste en calcular el número de caminos que le permiten

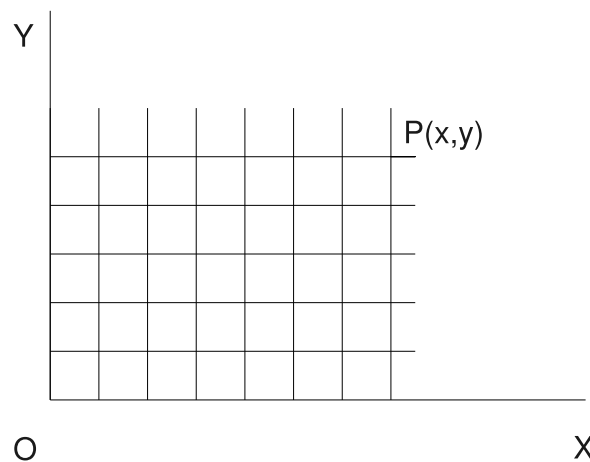


Figura 4.2: Plano de la ciudad.

acceder al punto P desde el origen O de la forma más rápida posible. Se puede seguir un razonamiento similar al del juego de la rayuela. El caso base o elemental se da cuando el automobilista se quiere desplazar desde el origen a un punto perteneciente al eje X o al eje Y , ya que en ese caso sólo hay un camino posible, que es el desplazamiento en línea recta. El caso general se formula sabiendo que a un punto $P(x, y)$ se puede acceder, de forma directa, sólo desde un punto $P_1(x - 1, y)$ o desde un punto $P_2(x, y - 1)$, ya que esos son los movimientos elementales.

- Caso elemental de la recursión:

$$\text{numeroCaminos}(x, 0) = 1$$

$$\text{numeroCaminos}(0, y) = 1$$

- Caso general:

$$\text{numeroCaminos}(x, y) = \text{numeroCaminos}(x, y-1) + \text{numeroCaminos}(x-1, y)$$

El algoritmo quedaría:

Algoritmo $Plano(x, y)$

inicio

si $x = 0$ **entonces**

devolver 1

sino

si $y = 0$ **entonces**

devolver 1

sino

devolver $Plano(x - 1, y) + Plano(x, y - 1)$

finsi

finsi

fin

4.4.3. Las Torres de Hanoi.

Según una antigua leyenda, cuando Dios creó el mundo, situó sobre la tierra 3 varillas de diamante y 64 discos de oro, cada uno de ellos de diferente tamaño y con un agujero en el centro, de forma tal que un disco se puede insertar en una varilla pasando ésta a ser el eje del disco. Inicialmente se insertaron todos los discos en la primera varilla en orden decreciente de diámetros, de forma tal que un disco siempre descansa sobre otro de tamaño superior. Diós les encargó a unos monjes trasladar, de uno en uno, todos los discos de la primera varilla a la segunda varilla, de forma tal que los discos sigan estando en orden decreciente de diámetros. Para ello debían usar la tercera varilla como ayuda, permitiendo mover un disco de una varilla a otra cualquiera siempre y cuando se colocase sobre un disco de mayor tamaño. Según dicha leyenda, cuando los monjes terminasen dicha tarea el mundo llegaría también a su fin. Aparentemente, el número de movimientos a realizar no sería excesivo, pero si se estudia el problema con detalle llegaríamos a la conclusión de que ésta es una de las profecías más optimistas sobre el fin del mundo, ya que suponiendo que cada disco tardase en moverse un segundo, y que los monjes trabajasen día y noche, aún quedarían algunos cientos de millones de años para que llegase el fin del mundo.

Para resolver el problema, se puede plantear la siguiente descomposición: supongamos que tenemos m discos, para trasladar éstos de la varilla 1 a la 2, se podrían trasladar los $m - 1$ discos superiores de la varilla 1 a la 3, después trasladar el disco m de la 1 a la 2 y por último, trasladar los $m - 1$ discos de la 3 a la 2. De esta forma, el procedimiento de trasladar m discos se ha descompuesto en dos procedimientos de traslado de $m - 1$ discos y un movimiento elemental de traslado de un disco. De la misma forma, cada procedimiento de traslado de $m - 1$ discos se podría descomponer en dos movimientos de traslado de $m - 2$ discos y otro movimiento

elemental de traslado de un disco. Siguiendo esta descomposición, podemos reducir a procedimientos correspondientes a los traslados elementales de un solo disco.

Es importante tener en cuenta que en los distintos procedimientos que van surgiendo, van cambiando la varilla origen, la de destino y la auxiliar. Para tener esto en cuenta, la varilla origen se puede designar con la variable i , la destino con la variable j y la auxiliar será la $6 - i - j$ donde $1 \leq i, j \leq 3, i \neq j$. En el momento inicial $i = 1, j = 2$, de esta forma las varillas irán intercambiando su misión a lo largo de las iteraciones.

El algoritmo quedaría de la siguiente forma:

En la primera llamada $i = 1, j = 2$

Algoritmo $Hanoi(m, i, j)$

inicio

si $m > 0$ **entonces**

$Hanoi(m - 1, i, 6 - i - j)$

escribir $i \rightarrow j$

$Hanoi(m - 1, 6 - i - j, j)$

finsi

fin

Para resolver el problema original bastaría con invocar al algoritmo con los parámetros $(64, 1, 2)$. Dado que cada llamada de orden m genera dos llamadas de orden $m - 1$, el número de llamadas crece exponencialmente y es del orden de 2^m . Por tanto, el algoritmo tiene una complejidad computacional $O(2^n)$.

4.5. Cuestiones sobre el tema

1. Explica en qué consiste un algoritmo recursivo.
2. Indica cual es la principal desventaja de los algoritmos recursivos. Justifica tu respuesta.
3. ¿Cual es el principal inconveniente de la versión recursiva del algoritmo de la sucesión de Fibonacci?. ¿Cómo se podría modificar dicha versión recursiva, sin eliminar la recursividad, para eliminar ese inconveniente?.
4. ¿Porqué la versión recursiva del algoritmo de Fibonacci consume más memoria que su versión iterativa?.
5. ¿Porqué la versión recursiva del cálculo del factorial de un número tarda más tiempo que la iterativa pese a tener la misma complejidad computacional?.
6. ¿Qué inconvenientes podría tener el uso de una tabla o de una estructura de datos adicional para evitar la repetición de llamadas en un algoritmo recursivo?

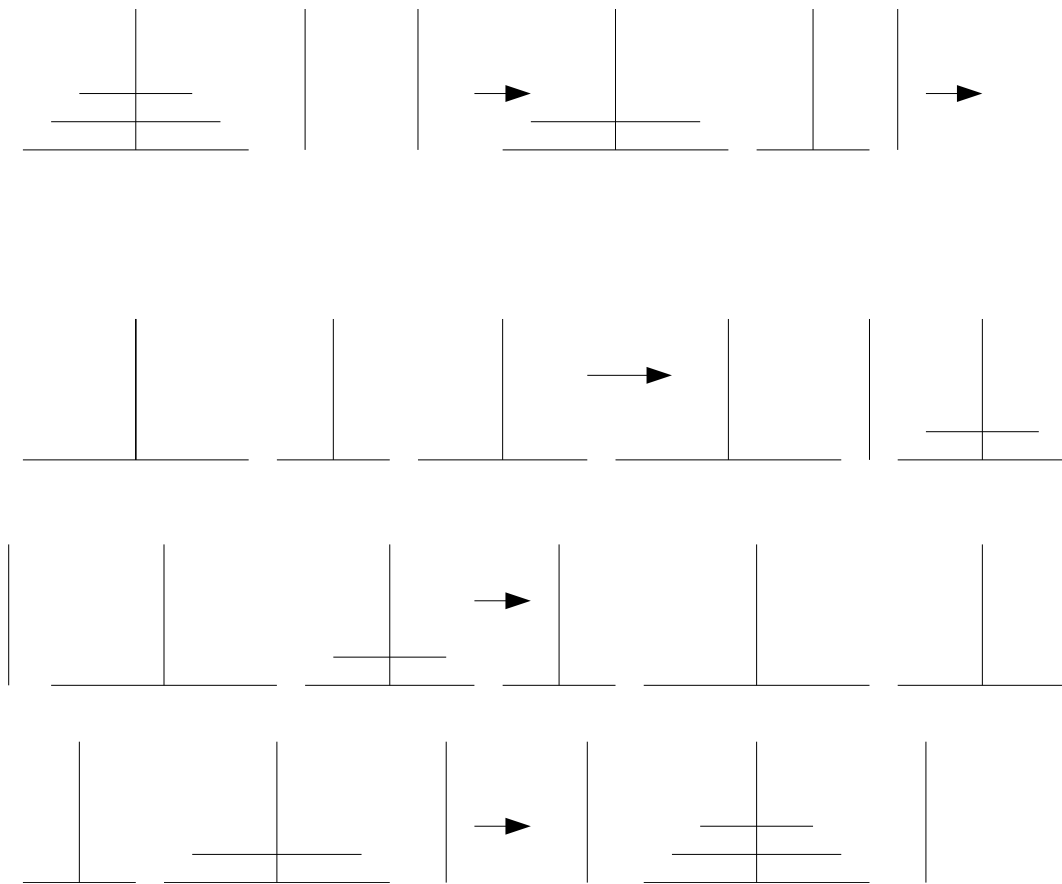


Figura 4.3: Resolución del problema de las torres de Hanoi para tres discos.

7. Indica cual es la principal ventaja de la recursividad. Cita al menos un par de algoritmos donde esa ventaja es patente.
8. ¿En qué casos valdría la pena traducir un algoritmo recursivo a uno iterativo?.
9. Si implementamos el cálculo del factorial de un número mediante un algoritmo iterativo y mediante un algoritmo recursivo. ¿Cual sería el orden de complejidad de cada implementación?. ¿Cual de los dos consumiría más tiempo?. Justifica esta última respuesta.
10. Si implementamos el cálculo del término n de la sucesión de Fibonacci mediante un algoritmo iterativo y mediante un algoritmo recursivo. ¿Cual sería el orden de complejidad de cada implementación?. ¿Cual de los dos consumiría más tiempo?. Justifica esta última respuesta.

11. Si implementamos el cálculo del término n de la sucesión de Fibonacci mediante un algoritmo iterativo y mediante un algoritmo recursivo que evitase la repetición de las llamadas recursivas. ¿Cual sería el orden de complejidad de cada implementación?. ¿Cual de los dos consumiría más tiempo?. Justifica esta última respuesta.
12. ¿De qué orden de complejidad es el algoritmo recursivo del juego de la rayuela?. ¿Se podría reducir esta complejidad y cómo se haría en caso de que fuese posible?.
13. ¿Qué ocurriría si en el juego de la rayuela se permitiese retroceder un adoquín?.
14. ¿Cómo sería los casos particulares y el caso general en el juego de la rayuela si se admitiese un movimiento para pasar del adoquín n al $n+3$?.
15. ¿De qué orden de complejidad es el algoritmo recursivo para obtener el término general de la sucesión de Fibonacci?. ¿Se podría reducir esta complejidad y cómo se haría en caso de que fuese posible?.
16. Indica la fórmula recursiva (caso particular y general) para el problema del plano de la ciudad si se permitieran movimientos de un solo paso a la derecha, hacia arriba o en diagonal noreste.
17. ¿De qué orden de complejidad es el algoritmo recursivo del plano de la ciudad?. ¿Se podría reducir esta complejidad y cómo se haría en caso de que fuese posible?.
18. ¿De qué orden de complejidad es el algoritmo de las torres de Hanoi?. Justifica tu respuesta.
19. ¿Se podría utilizar una tabla para evitar llamadas repetidas en el algoritmo de las torres de Hanoi para reducir su tiempo de ejecución?. Justifica tu respuesta.
20. ¿Qué recurso se utiliza para numerar las varillas en el algoritmo de las torres de Hanoi?