Proyecto 1: Torres de Hanoi

Juan M. Alberola, Jose I. Pastor

jalberola@dsic.upv.es

Grado en Tecnologías Interactivas

2 sesiones

Introducción

El problema de las **Torres de Hanoi** es un puzzle matemático que consiste en mover n discos de diferente tamaño de un palo o pivote inicial a un palo final, utilizando un palo auxiliar (Figura 1).

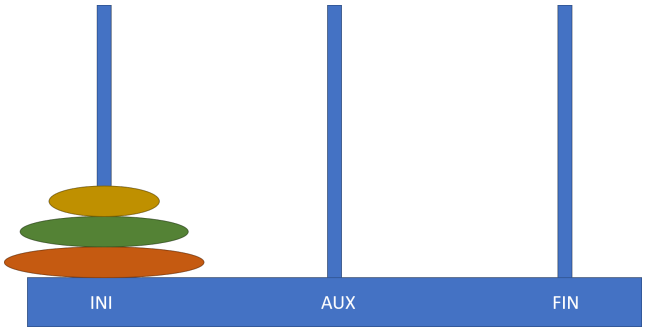

Figura 1: Torres de Hanoi con 3 discos

El movimiento de los discos se rige por una serie de reglas:

- Sólo puede moverse un disco en cada movimiento.
- Un disco puede moverse solo si no tiene ningún disco encima, es decir, si es el que está en la parte superior de la pila de discos.
- Ningún disco puede situarse encima de otro disco más pequeño.

Tal y como habéis visto en la parte teórica de la asignatura, la cantidad de posiciones posibles que se pueden tener con n discos se representa como las **variaciones con repetición** de los 3 palos y los n discos, es decir:

$$Pos(n) = VR_{3,n} = 3^n$$

Para solucionar el problema, se pueden seguir distintos tipos de aproximaciones, pero está demostrado matemáticamente que el mínimo número de movimientos que se requiere para solucionar un problema con n discos es $2^n - 1$.

Descripción del proyecto

En este proyecto deberás implementar una aplicación de consola en **C#** con **Visual Studio** a partir del código inicial que se proporciona en el siguiente repositorio:

https://gitlab.com/Alberola/torres-de-hanoi.git

Como puedes ver, este proyecto consta de diferentes clases, que pasamos a detallar a continuación.

Clase Disco

Esta clase representa un objeto individual disco. Como podemos ver en el ejemplo de la Figura 1, cada disco se diferencia de otro por su tamaño. Para representar esto en el código, podemos utilizar valores enteros o cadenas de caracteres, según tu criterio.

Clase Pila

Esta clase representa un objeto individual palo junto con los discos que hay en ese palo. Para entender las propiedades y métodos de esta clase, vamos a analizar los movimientos que podemos hacer con los discos.

El movimiento de extraer un disco de un palo, consistirá en quitar el disco que está en la parte superior. Tal y como podemos ver en la Figura 2, si extraemos un disco del palo *INI*, el disco que extraeremos será el amarillo.

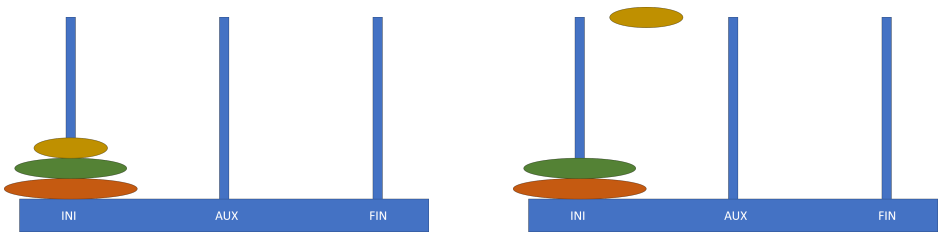

Figura 2: Movimiento de extracción de un disco del palo *INI*

Por otro lado, el movimiento de colocar un disco en un palo, siempre lo colocará sobre el último disco que haya en ese palo (Figura 3 o bien sobre la base, si es que no hay ninguno (Figura 4).

Este tipo de movimientos, los podemos representar con una estructura de datos llamada **pila**. Una pila, consiste en un conjunto de elementos (en este caso, discos), en donde el primer elemento que se puede extraer es el último que se ha insertado. Para representar esto, necesitamos definir las siguientes propiedades:

- `Size` para representar la cantidad de discos que hay en el palo.

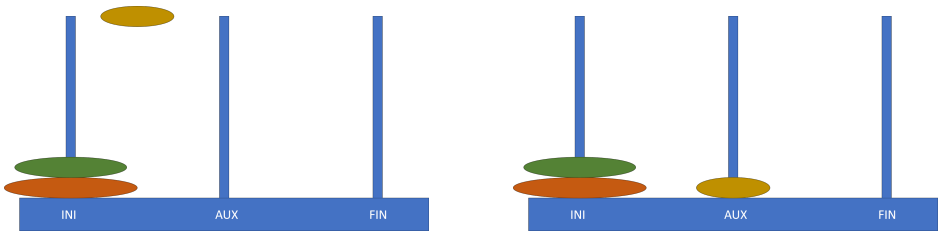

Figura 3: Movimiento de colocar de un disco en el palo *INI*

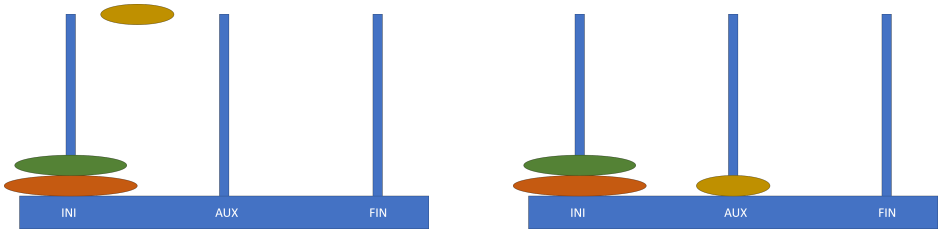

Figura 4: Movimiento de colocar de un disco en el palo *AUX*

- `Top` para representar el disco que está en la parte superior del palo.
- `Elementos` para representar el conjunto de discos que hay en ese palo.

Además, necesitamos los siguientes métodos:

- `public Pila()` que representa el constructor de la clase.
- `public void push(Disco d)` que permitirá colocar un disco en la pila de discos.
- `public Disco pop()` que permitirá extraer el disco de la parte superior de la pila.
- `public bool isEmpty()` que informará si la pila está o no vacía.

Hanoi

Esta clase representa la lógica de los movimientos y la resolución del problema.
En primer lugar, necesitamos un método que nos permita mover un disco entre dos palos:

```
public mover_disco(Pila a, Pila b)
```

Ten en cuenta que dados dos palos a y b, sólo habrá un posible movimiento posible:

- Extraer un disco de a y colocarlo en b siempre que el disco de arriba del palo a sea más pequeño que el que hay en la parte superior de b.
- Extraer un disco de b y colocarlo en a siempre que el disco de arriba del palo b sea más pequeño que el que hay en la parte superior de a.

Por tanto, este método deberá comprobar qué movimiento es el válido, y posteriormente extraer el disco del palo correspondiente y colocarlo en el otro palo.

En segundo lugar, necesitamos un método que resuelva el problema de las Torres de Hanoi en el número mínimo de movimientos. Una solución simple a este problema, consiste en un **algoritmo iterativo** que repite tres tipos de movimientos hasta llegar a la solución. Estos tres tipos de movimientos serán diferentes dependiendo de si tenemos un número par o impar de discos. El algoritmo iterativo debe devolver la cantidad m de movimientos necesarios para mover n discos de un palo INI a un palo FIN utilizando un palo AUX. El pseudocódigo de este algoritmo iterativo se puede ver a continuación:

Algoritmo iterativo

```
ENTRADA: n, INI, FIN, AUX
SALIDA: m
si n es impar:
    mientras no haya solución:
        incrementar_movimientos(m) ← mover_disco(INI, FIN)
        incrementar_movimientos(m) ← mover_disco(INI, AUX)
        incrementar_movimientos(m) ← mover_disco(AUX, FIN)
    fin mientras
fin si
si n es par:
    mientras no haya solución:
        incrementar_movimientos(m) ← mover_disco(INI, AUX)
        incrementar_movimientos(m) ← mover_disco(INI, FIN)
        incrementar_movimientos(m) ← mover_disco(AUX, FIN)
    fin mientras
fin si
devuelve m
```

Como puedes ver en el algoritmo, dependiendo de si el número de discos es par o impar, se seguirán tres movimientos hasta encontrar la solución. Hay que tener en cuenta que la solución se puede después de realizar **cualquiera** de estos tres movimientos. Podemos definir la solución como un escenario en donde todos los discos estén en el palo *FIN*. Es interesante que incluyas mensajes que muestren por consola el movimiento que se realiza.

Program

Esta es la clase principal de tu aplicación. En primer lugar, deberás inicializar los tres palos. Ten en cuenta, que el palo representado como *INI* tendrá los n discos mientras que los palos *FIN* y AUX estarán vacíos.

Después, deberás realizar la llamada al algoritmo iterativo que soluciona el problema, mostrando el número de movimientos realizados y comprobando que se corresponde con el número mínimo de movimientos requeridos.

Pasos a seguir

En este proyecto, deberás implementar las clases indicadas:

1. Implementar la clase *Disco*, decidiendo el tipo para diferenciar un disco de otro.
2. Implementar la clase *Pila*, decidiendo cómo representamos la pila de discos. Además, deberemos implementar los distintos métodos.
3. Implementar los métodos de la clase *Hanoi*.
4. Implementar la clase principal *Program*.

Parte opcional para llegar a la máxima nota

La solución del problema de las Torres de Hanoi también puede aproximarse utilizando el concepto de **recursión**. Un **algoritmo recursivo** resuelve un problema realizando una llamada a sí mismo. Esto se puede ver mejor con un ejemplo.

Imaginemos que queremos calcular el **factorial** de un número n. Una posible solución a este problema podría ser calcular el factorial de $n-1$ y multiplicarlo por n. A su vez, el factorial de $n-1$ podría resolverse calculando el factorial de $n-2$ y multiplicando la solución por $n-1$. Evidentemente, las llamadas recursivas tienen que parar en algún momento, y esta condición de parada se le llama **caso base**.

Un caso base representa una solución al problema que puede resolverse de manera simple sin utilizar recursión. En el caso del problema del factorial, el caso base puede representarse para $n = 0$, en donde $factorial(n) = 1$. Por tanto, podemos definir el caso base y el caso general de la recursión de la siguiente manera:

$$factorial(n) = \begin{cases} 1 & \text{si } n = 0 \\ factorial(n-1) \times n & \text{si } n > 0 \end{cases} \quad (1)$$

En el caso de las Torres de Hanoi, la solución recursiva para un problema con 3 discos del palo *INI* al palo *FIN*, consistirá en resolver el problema de mover los dos discos más pequeños del palo *INI* al palo *AUX*, después mover el disco grande al palo *FIN*, y finalmente, mover los dos pequeños del palo *AUX* al palo *FIN*.

A su vez, el problema de mover dos discos del palo *INI* al palo *AUX*, consiste en resolver el problema de mover el disco pequeño al disco *FIN*, después, mover el disco más grande al palo *AUX*, y finalmente, mover el disco pequeño del palo *FIN* al palo *AUX*. Este proceso sería similar también para el problema de mover los dos pequeños del palo *AUX* al palo *FIN*.

Podemos ver que en este problema, el caso base consiste en mover un disco de un palo a otro, que simplemente incluye un movimiento de extracción y otro de colocación. Teniendo en cuenta esto, el pseudocódigo de este algoritmo recursivo se puede ver a continuación:

Algoritmo recursivo

ENTRADA: n, INI, FIN, AUX
SALIDA: m
si $n = 1$:
    $extraer\_disco(INI)$
    $colocar\_disco(FIN)$
    $m \leftarrow m + 1$
sino:
    $incrementar\_movimientos(m) \leftarrow algoritmo\_recursivo(n-1, INI, AUX, FIN)$
    $extraer\_disco(INI)$
    $colocar\_disco(FIN)$
    $m \leftarrow m + 1$
    $incrementar\_movimientos(m) \leftarrow algoritmo\_recursivo(n-1, AUX, FIN, INI)$
fin si
devuelve m

Entrega y exposición

Se realizará la entrega del proyecto en la plataforma dentro de la fecha límite. Además, se tendrá que mostrar el correcto funcionamiento en la sesión indicada por el profesor, el cuál realizará preguntas y podrá proponer modificaciones para comprobar que se entiende.

Será requisito también, enseñar el repositorio local de Git con almenos, los siguientes commits:

- Implementada clase Disco.
- Implementada clase Pila.
- Implementado método de mover disco.
- Implementado algoritmo iterativo.

Criterios de evaluación

- Funcionamiento completo incluyendo opcional: 10.
- Funcionamiento completo sin incluir opcional: 8.
- Conoce y expone los pasos realizados de una forma clara y estructurada: sin penalización.
- Conoce los pasos pero su exposición no es clara ni estructurada: hasta 2 puntos menos.
- Hay partes que no entiende o no explica correctamente: hasta 5 puntos menos.
- No expone: entre 5 y 10 puntos menos.
- Realiza modificaciones con seguridad y claridad: sin penalización.
- Tiene dudas con algunas modificaciones: hasta 2 puntos menos.
- Tiene dudas con todas las modificaciones planteadas: hasta 5 puntos menos.

- Hay algún error leve: entre 0 y 2 puntos menos.
- Hay algún error grave: entre 1 y 5 puntos puntos.
- No se expone ni se entrega: 0 puntos.
- Cualquier modificación de estos criterios, se informará con suficiente antelación