Capítulo 2

|  |  |  |
| --- | --- | --- |
| Jonathan Zinzan Salisbury Vega  *Universitat de les Illes Balears* Palma, España jonathan.salisbury1@estudiant.uib.cat | Joan Sansó Pericás  *Universitat de les Illes Balears* Palma, España joan.sanso4@estudiant.uib.cat | Joan Vilella Candia  *Universitat de les Illes Balears* Palma, España joan.vilella1@estudiant.uib.cat |

*El coste asintótico de los algoritmos ha sido uno de los principales focos de estudio (y de problemas) a lo largo de la historia de la informática. Las soluciones más sencillas suelen tener costes asintóticos no deseables dependiendo del problema a resolver. Además, encontrar soluciones óptimas suele traer verdaderos quebraderos de cabeza a los distintos programadores. En este proyecto se ha implementado una aplicación capaz de visualizar los distintos tiempos que tienen estos algoritmos. Para ello se ha utilizado el Modelo Vista Controlador. Utilizar esta arquitectura permite un mayor control de errores del programa y facilidad de reutilización como de escalado. Todo esto a coste de una mayor complejidad de código. Con esta práctica se busca poner en práctica la arquitectura MVC y ofrecer una herramienta al programador para poder discernir de una manera gráfica con la que estudiar la viabilidad de los distintos algoritmos.*

Keywords—MVC, rendimiento, coste asintótico, algoritmo.

# Introducción

Una de las máximas que ha de tener un programador a la hora de diseñar sus algoritmos, es el coste asintótico de los mismos. Muchas veces, por un simple descuido o por desconocimiento sobre la materia, se acaban implementando soluciones muy alejadas de lo que podríamos considerar como aceptable.

La herramienta que se ha desarrollado permite visualizar de una manera más fácil el coste asintótico de los distintos órdenes de complejidad. Esto permite al usuario hacerse una idea de la viabilidad de sus soluciones, no necesariamente para que sean las óptimas, pero sí para que se puedan realizar en un tiempo aceptable.

El segundo objetivo de la práctica ha sido la puesta en práctica del MVC (Modelo Vista Controlador) visto en clase. La estructura del documento será la siguiente, en primer lugar, se describe la implementación del modelo MVC y después se comentan los resultados de los distintos costes asintóticos.

# Modelo vista controlador

Antes de explicar el modelo vista controlador, es importante entender qué es un patrón de diseño (o arquitectura).

Un patrón de diseño es un conjunto de técnicas que se estandarizan para resolver un conjunto de problemas comunes en el desarrollo de software. No toda técnica puede ser considerada un patrón de diseño, por lo que debe cumplir las siguientes características:

* Efectividad ante problemas similares.
* Reutilizable

Por lo que un patrón podría ser considerado una plantilla en la que se unifican una serie de buenas prácticas. El Modelo Vista Controlador no es más que un patrón de entre todos los que hay. Pero recoge una serie de particularidades especialmente útiles para el diseño de esta práctica.

El consenso general ha dividido los patrones de diseño en tres grandes grupos.

* Creacionales
* Estructurales
* Comportamiento

Los creacionales están enfocados en la producción de objetos bajo una serie de criterios. Los estructurales se encargan en la organización de clases y objetos para brindar una solución y finalmente, los de comportamiento identifican similitudes entre la interacción de clases y objetos. Además, existe una distinción para los patrones con soluciones basadas en concurrencia.

Basándonos en las definiciones antes dadas, es difícil colocar al MVC en uno de los tres grupos. Esto es porque este patrón es de “dominio específico”, por lo que está muy orientado a las interfaces de usuario. Dentro de este grupo también se pueden encontrar soluciones específicas de: la seguridad informática, modelos de negocios, entre otros.

## Historia y definición original

El MVC lo crea Trygve Reenskaug, ingeniero informático y profesor emérito en la Universidad de Oslo, durante el desarrollo del lenguaje Smalltalk-79, a finales de los 70. Después de varios diseños, opta por el actualmente conocido MVC.

En propias palabras de Trygve Reenskaug: *“El propósito esencial de MVC es cerrar la brecha entre el modelo mental del usuario humano y el modelo digital que existe en la computadora. La solución MVC ideal respalda la ilusión del usuario de ver y manipular la información del dominio directamente. La estructura es útil si el usuario necesita ver el mismo elemento del modelo simultáneamente en diferentes contextos y/o desde diferentes puntos de vista. La siguiente figura ilustra la idea.”*

Diagram

Description automatically generated

Como se puede observar en la definición original, el objetivo primordial de este modelo es facilitar el entendimiento al usuario final (modelo mental), facilitando así la conexión con el modelo. Es importante tener esto en cuenta porque muchas veces se desvirtúa la intención de este patrón, pensando que es una composición elegante de tres instancias.

## Definición actual y ventajas

La definición más aceptada hoy en día es la propuesta por el libro Design Patterns: Elements as Reusable Object-Oriented Software (1994):

“*MVC consta de tres tipos de objetos. El modelo es el objeto de la aplicación, la vista es su presentación en pantalla y el controlador define la forma en que la interfaz de usuario reacciona a la entrada del usuario. Antes de MVC, los diseños de interfaz de usuario tendían a agrupar estos objetos. MVC los desacopla para aumentar la flexibilidad y la reutilización.*”

Como podemos observar, en esta definición ya se introduce el concepto de objeto, el cual es primordial en la programación actual. Además, ofrece una aproximación más actualizada de la función de cada una de las partes del patrón MVC.

Otra de las características más importantes, que no se mencionan en las anteriores definiciones, es el establecimiento de un protocolo suscripción/notificación entre los tres objetos. Es decir, cada vez que los datos del modelo cambien, el modelo deberá notificar a todas las vistas que dependen de él. En respuesta, las vistas deberán de actualizarse. Esto permite tener múltiples vistas adjuntadas a un modelo, en el que se podrán hacer diferentes presentaciones de la información.

Recapitulando todas las ventajas clave del patrón:

* Capacidad de adjuntar múltiples vistas para un modelo, permitiendo así distintas representaciones.
* Facilidad a la hora de cambiar la forma en que una vista responde a la entrada del usuario sin cambiar su presentación visual.
* Reutilización de código.

Ahora bien, este patrón añade una complejidad a la hora de programar. Ya que se debe realizar una separación en tres objetos a la hora de plantear una solución al problema. Si no se necesita realizar este desacoplamiento en tres objetos, se está optando por una complejidad innecesaria.

## Contexto de estudio

# Recursividad

La recursividad es la forma en la cual un proceso se especifica basado en su misma definición. Más concretamente, en programación, es un método usual de resolver un problema de forma sencilla, como puede ser el problema de las [Torres de Hanoi](https://www.geeksforgeeks.org/c-program-for-tower-of-hanoi/), recorrido de Grafos, la búsqueda Euleriana…

Otros ejemplos de algoritmos que se definen de forma recursiva por su naturaleza son: los [Números de Fibonacci](https://en.wikipedia.org/wiki/Fibonacci_number), la [función factorial](https://en.wikipedia.org/wiki/Factorial).

A continuación, vamos a explicar los distintos tipos de recursividad.

## Tipos de recursividad

Dependiendo de la estructura de una función recursiva puede ser de los siguientes tipos:

### Recursividad Directa

La recursividad directa es cuando la función se llama a sí misma, pero tanto antes de la llamada recursiva como al volver, hay código que se ejecuta.

### Recursividad Indirecta

Se llama recursividad Indirecta cuando una función no se llama a sí misma, sino que esta desencadena una serie de llamadas de funciones, donde en una de estas llamadas se llama a la función original.

### Recursividad de cola

La recursividad de cola es un tipo de recursividad directa donde la llamada recursiva es la última sentencia de la función (todas las operaciones se hacen antes de la llamada recursiva).

### Recursividad de cabeza

Como la recursividad de cola, es un tipo de recursividad directa donde la llamada es la primera sentencia de la recursión. Es decir, las operaciones se realizan cuando la recursión ha llegado al caso base y las llamadas recursivas empiezan a volver.

### Recursividad Lineal

Es un tipo de recursión donde la cantidad de llamadas es proporcional al tamaño del problema (solo hay una llamada recursiva en la definición de la función, p. ej.: Factorial).

### Recursividad en Árbol o no Lineal

A diferencia de la Lineal, en estas funciones se hace más de una llamada recursiva, y, por ende, la recursión crece en forma de árbol (p ej.: Fibonacci, donde se hacen 2 llamadas recursivas en la definición de la función).

## Ventajas e inconvenientes de la Recursividad vs los algoritmos iterativos

### Ventajas

La recursividad puede **reducir la complejidad temporal**. Esto puede parecer contraintuitivo, pero si aplicas programación dinámica y **memorizas** soluciones parciales, puedes disminuir la complejidad temporal. Por ejemplo, para el cálculo de los números de Fibonacci, donde el algoritmo que memoriza es simplemente de complejidad tanto temporal como espacialmente (si se usa la estructura de datos adecuada).

Los programas recursivos suelen ser más “simples” y suelen tener menos líneas de código.

Son muy buenos para recorrer árboles. Debido a esto, se usan para implementar recorridos de árboles como el Preorden y para la generación de éstos (*véase Backtracking*).

### Desventajas

Si la cantidad de llamadas seguidas es muy grande puede dar problemas de memoria, ya que las llamadas van llenando la pila del sistema, y puede haber un error de *Overflow*, que será aún más rápido si se usan muchas variables locales, ya que en cada llamada recursiva éstas se tienen que guardar, ocupando aún más espacio.

Si se implementa mal, o se usa la recursividad en problemas sencillos, puede ser muy lenta. Esto es debido a que la recursividad necesita más tiempo al hacer llamadas y volver de estas debido a todo el manejo de memoria que se tiene que hacer, mientras que las iterativas son simples bucles sobre los mismos datos.

# Backtracking

El *backtracking* es un algoritmo general para encontrar soluciones a problemas que tienen que satisfacer un conjunto de condiciones.

La traducción literal del nombre seria “Paso atrás”, y se le da ese nombre debido a la naturaleza de estos algoritmos: van creando candidatos a la solución de manera incremental, y abandonan un candidato (dan un paso atrás) cuando se determina que ese candidato no podría ser solución.

De esta manera, se va generando un *árbol de soluciones parciales*, y el algoritmo **poda** la rama del árbol cuando ve que la solución parcial no podrá llegar a una solución completa.

El algoritmo puede ser programado para que pare al encontrar una solución, o puede ser creado de forma que encuentre todas las soluciones, que pare cuando haya pasado un tiempo determinado, o haya gastado una cantidad de recursos (ciclos de CPU, memoria) determinados.

## Ventajas e inconvenientes del Backtracking

Al ser un subconjunto de los algoritmos recursivos, se aplican las ventajas y desventajas de éstos. Adicionalmente, podemos enumerar:

### Ventajas

El backtracking, al ser un algoritmo que recorre todo el árbol de posibilidades, siempre va a ser capaz de encontrar todas solución, si es que existe alguna. De la misma manera, es capaz de asegurarte que no existe ninguna solución al conjunto de restricciones o condiciones establecidos.

Para según que tipos de problemas, es mucho más sencillo programar un algoritmo recursivo que intentar hacer uno iterativo para el mismo objetivo. Esto es debido a la facilidad de aplicar el concepto de recursión en problemas que necesitan satisfacer una lista arbitraria de condiciones, ya que puedes hacer que cada llamada recursiva intente encontrar una solución parcial a una de las condiciones, y si llegas a un estado que no podrá desarrollarse en solución final, vuelves atrás e intentas de nuevo.

### Inconvenientes

El coste asintótico de estas funciones suele ser del orden factorial () o exponencial (). Esto hace que para según qué problemas de gran tamaño, sea impracticable en el tiempo.

# Recorrido euleriano

Para entender que es un gráfico euleriano se intentará resolver un problema típico. Dado un grafo, ¿Se puede dibujar sin la necesidad de levantar el lápiz y sin repetir líneas? En caso de ser así, estamos ante un grafo euleriano, por lo que el camino euleriano será el conjunto de nodos recorridos en ese orden.

Una vez entendido a nivel informal la definición de recorrido euleriano, dará una definición matemática.

“Un ciclo o circuito euleriano en la [Teoría de Grafos](https://www.ecured.cu/Teor%C3%ADa_de_Grafos) es aquel camino que recorre todas las aristas de un grafo pasando una y sólo una vez por cada arco (arista) del grafo, siendo condición necesaria que regrese al vértice inicial de salida (ciclo = camino en un grafo donde coinciden vértice inicial o de salida y vértice final o meta). Es aquel ciclo que contiene todas las aristas de un grafo solamente una vez.”

Es importante hacer una pequeña salvedad, en la realización de esta práctica se busca un recorrido euleriano, no un ciclo. La única diferencia es que un recorrido no tiene porqué acabar en el nodo inicial, en cambio, para un ciclo es una condición necesaria. Además, un grafo dibujado en un tablero de ajedrez es un grafo no dirigido.

A continuación, se mencionarán las distintas propiedades que tienen los recorridos eulerianos. Dichas propiedades permiten discernir a nivel matemático si se cumple el algoritmo y no simplemente realizar una justificación visual.

* Un grafo conexo y no dirigido se dice que es euleriano si cada vértice tiene un grado par.
* Un grafo no dirigido es euleriano si es conexo y se puede descomponer en uno con los vértices disjuntos.
* Si un grafo no dirigido G es euleriano entonces su grafo-línea L (G) se dice que es también euleriano.
* Un grafo dirigido es euleriano si es conexo y cada vértice tiene grados internos iguales a los externos.
* Un grafo no dirigido se dice que es susceptible de ser recorrido si es conexo y al menos dos vértices en el grafo tienen grado impar.

# descripción del problema

El problema que se plantea en esta práctica es una extrapolación del conocido juego “El salto del caballo”. En el juego original, se ha de encontrar un recorrido (sin repetir casillas) desde una casilla inicial a otra casilla final, sin pasar por una ya visitada.

Como es fácil de observar, la descripción del problema encaja a la perfección con la definición del recorrido euleriano. Las casillas del tablero corresponderán con los nodos del grafo y las aristas serán los distintos movimientos que harán las piezas.

# Bibliografía

|  |  |
| --- | --- |
| [1] | Oracle, «UISwing Concurrency - Swing Worker,» [En línea]. Available: https://docs.oracle.com/javase/tutorial/uiswing/concurrency/worker.html. |
| [2] | Wikipedia, «Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Divide-and-conquer\_algorithm. |
| [3] | «Oracle Java Documentation - SwingWorker,» [En línea]. Available: https://docs.oracle.com/javase/7/docs/api/javax/swing/SwingWorker.html. |
| [4] | J. M. Alarcón, «Rendimiento de Algoritmos y Big O - Campus MVP,» [En línea]. Available: https://www.campusmvp.es/recursos/post/Rendimiento-de-algoritmos-y-notacion-Big-O.aspx. |
| [5] | P. M. H. Thomas V. Perneger, «Writing a research article: advice to beginners - Oxford Academic,» [En línea]. Available: https://academic.oup.com/intqhc/article/16/3/191/1814554. |
| [6] | Wikipedia, «Recursion - Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Recursion. |
| [7] | Wikipedia, «Backtracking - Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Backtracking. |