Capítulo X

|  |  |  |
| --- | --- | --- |
| 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 |

Mini Resumen

Keywords—

# Introducion

En este documento se pretende explicar…

# Contexto y entrono de estudio

Uno de los requisitos a la hora de realizar la práctica es el uso del lenguaje de programación Java. Además, se ha dado la opción de elegir entre dos IDE (Integrated Development Environment).

* NetBeans
* IntelliJ

En este caso se ha escogido el IDE de NetBeans por familiaridad de uso. Además, se utiliza una herramienta de control de versiones (Git). Más específicamente su versión de escritorio Github Desktop por su facilidad de uso mediante interfaz gráfica.

# Despcripcion del problema

El problema que se plantea en esta práctica es el desarrollo y comparación de tres algoritmos de multiplicación de dos enteros. Estos han de ser de N y M cifras, por lo que hay que tener en cuenta la longitud de estos.

Los algoritmos que se han implementado en esta práctica son:

* Tradicional
* Karatsuba
* Mixto

## Multiplicacion Tradicional

Es importante entender la connotación de “tradicional” a la hora de analizar el funcionamiento de este algoritmo.

*“Los algoritmos convencionales tratan a las cifras en forma aislada como si fuesen números y no se tiene noción de la totalidad que implican las cifras, es decir el valor que tienen por su posicionalidad en el numeral. Además, ocultan cálculos y propiedades que se aplican.”*

El funcionamiento del algoritmo consiste en ir de derecha a izquierda multiplicando teniendo en cuenta: ley de signos y orden de unidades. Para ello se colocarán unidades debajo de mismo orden de unidades, finalmente se suman los productos de cada cifra del segundo factor por todas las del primero. La suma se realiza también de manera tradicional, de derecha a izquierda.

Table, Word

Description automatically generated

## Algoritmo de Karatsuba

Este algoritmo es un algoritmo de multiplicación rápida. Esto se debe a que logra reducir la multiplicación de n dígitos a como máximo **FORMULA**. Esto se consideraba imposible hasta que Anatolii Alexeevitch Karatsuba lo descubrió en el año 1960. Permitiendo así reducir el algoritmo clásico que necesita **FORMULA** productos por dígito.

A continuación, se explicará el funcionamiento del algoritmo, la demostración matemática queda fuera del alcance de este documento.

El algoritmo se fundamenta en el uso de la siguiente ecuación:

Donde N es el número original. y son las subcadenas en las que se descompone el número original y m es la base del número.

Teniendo en cuenta esta fórmula, se desarrollan las siguientes teniendo en cuenta un producto de dos números x e y.

Si se aplican las siguientes igualdades:

Ahora bien, para esto se necesitan cuatro multiplicaciones. Karatsuba se dio cuenta que se puede calcular en función de z\_2 y z\_0. Reduciendo así el número de multiplicaciones a costa de unas restas. Por lo que z\_1 quedaría:

Finalmente, la fórmula sería:

## Algoritmo Mixto

(Explicar Divide and conquer) [Divide-and-conquer algorithm - Wikipedia](https://en.wikipedia.org/wiki/Divide-and-conquer_algorithm#:~:text=A%20divide%2Dand%2Dconquer%20algorithm,solution%20to%20the%20original%20problem.)

# Solucion propuesta

## Algoritmo Tradicional

## Karatsuba Mediante Recursividad

### Que es la recursivdidad, pq la usamos y como la usamos

### Solucion Karatsuba

### Solucion Mixto

## Complejidad de la solucion

# Patron MVC

El Modelo-Vista-Controlador (MVC) [1] es un patrón de diseño de software[[1]](#footnote-2) en el que se divide la lógica del programa de su representación gráfica, además se hace uso de un controlador para los eventos y comunicaciones entre las distintas partes.

Este patrón de arquitectura se basa en las ideas de reutilización de código y la separación de conceptos distintos, estas características pretenden facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento. Para ello se proponen la construcción de tres componentes:

* *Modelo*

Es donde se almacena la información necesaria para la ejecución de la aplicación. Gestiona el acceso a dicha información mediante peticiones a través del controlador. También contiene los procedimientos lógicos que hagan uso de esa información.

* *Vista*

Presenta la información en un formato adecuado para la interacción del usuario con los datos.

* *Controlador*

Responde a eventos usualmente generados por el usuario y los convierte en comandos para la vista o modelo. Se podría decir que hace la función de intermediario entre la vista y el modelo encargándose de la lógica del programa.

Como se ha mencionado anteriormente se ha decidido utilizar este patrón debido a la facilidad que aporta al programar la separación de conceptos. Además, se obtiene una capa más de abstracción ya que es posible utilizar diferentes vistas con un mismo modelo.

Hoy en día, es muy común el uso de la programación orientada a objetos (POO) como paradigma principal, por ello cada componente del patrón MVC suele implementarse como una clase independiente. A continuación, se explica que proceso se ha seguido y como se ha implementado cada parte.

## Implementacion Modelo

Se ha diseñado la clase *Model* que se encarga de almacenar todos los algoritmos y funciones auxiliares necesarias para resolver el problema.

Los algoritmos han sido diseñados para realizar todos los cálculos con el uso de *Strings* para poder almacenar números grandes sin ningún problema. Únicamente se utilizan valores numéricos para almacenar dígitos sueltos, nunca resultados compuestos. También cabe destacar que todos los algoritmos se han diseñado para que sean compatibles con el conjunto de números enteros *Z,* tanto con signo positivo como con negativo.

Los métodos para los principales algoritmos utilizan únicamente funciones para el tratamiento de Strings, y las funciones auxiliares *add()* y *sub()*. Estas funciones realizan las operaciones básicas de sumas y restas, pero utilizando *Strings* para que sea compatible con números grandes. Para que estas sean compatibles con números con signos se ha seguido el siguiente proceso:

* *Add()*:

|  |  |  |  |
| --- | --- | --- | --- |
| Num1 | Num2 | Res |  |
| +A | +B | +A+(+B) = A + B | add(A, B) |
| +A | -B | +A+(-B) = A - B | sub(A, B) |
| -A | +B | -A+(+B) = -A + B = B - A | sub(B, A) |
| -A | -B | -A+(-B) = -(A + B) | -add(A, B) |

Tabla

De esta manera conseguimos transformar todas las operaciones a operaciones normales sin signo. De esta forma dentro de la suma quedaría una suma normal, una resta normal, una resta girada y una suma añadiendo el menos.

* *Sub()*:

En este caso quedan las mismas operaciones, pero cambiando el orden de estas.

Además, hay definidas algunas funciones auxiliares para poder comparar los distintos algoritmos.

## Implementacion Vista

## Implementacion Controlador

### Concurrencia

# Guia de usuario

# Estudio de rendimiento y comparacion de los algoritmos

## Estudio de rendimiento

En este apartado se explicará el proceso que se ha llevado a cabo para hacer un estudio de rendimiento del ordenador de uno de los integrantes del equipo.

Para obtener los datos de rendimiento, se ha implementado un apartado en el programa que calcula el tiempo de cada algoritmo con un numero de entrada de tamaño N determinado.

Una vez hechos los cálculos, se guardan los resultados en un archivo externo (se hace después para no interrumpir el proceso de multiplicación).

Antes de pasar a los datos, se pasará a analizar cómo podemos obtener una medida que indique de alguna forma el rendimiento del ordenador.

Los datos provienen de dos algoritmos distintos: El algoritmo de multiplicación clásica, y el algoritmo de Karatsuba. Éstos tienen las siguientes fórmulas de complejidad asintótica:

Tradicional:

Karatsuba:

La medida que se busca es esa CM (Constante multiplicativa), que variará dependiendo del dispositivo.

Al saber la fórmula de complejidad asintótica, se puede afirmar que:

Por lo tanto, si se sabe el tiempo que ha tardado el algoritmo para una entrada N, se puede saber la Constante Multiplicativa para un algoritmo en concreto.

Una vez tenemos esto en mente, procedemos a analizar los datos obtenidos por los distintos algoritmos.

Se han ejecutado una serie de multiplicaciones con ambos algoritmos para un rango de N desde 1 hasta 700. Con esto, se espera tener una cantidad de datos significativa con la que obtener la medida deseada.

Se ha guardado en un archivo de texto los tiempos de cada N para cada algoritmo, y luego se han importado en una hoja de cálculo para el análisis.

Aquí podemos observar que el tiempo aumenta con respecto a N, y llega un momento en el que Karatsuba pasa a ser más rápido que el algoritmo tradicional.

Con las fórmulas descritas anteriormente, hemos calculado la constante multiplicativa para cada N, y hemos obtenido los siguientes resultados:

Como se puede observar, al inicio los valores son muy grandes, pero se estabilizan a partir de N ≈ 250. Por esto, vamos a quitar estos *outliers* para poder observar mejor los datos.

Aquí se puede apreciar mucho mejor como se va estabilizando la constante multiplicativa, donde se estabiliza para el algoritmo tradicional sobre 164 y para el Karatsuba sobre 506. Aplicamos una Media Móvil para suavizar y que se vea mejor:

Podemos afirmar que los valores se estabilizan sobre

# Conclusiones

# Bibliografía

[MATEMÁTICA EN LA ESCUELA PRIMARIA: LOS ALGORITMOS DE LAS OPERACIONES (mariamatica.blogspot.com)](http://mariamatica.blogspot.com/p/los-algoritmos-de-las-operaciones.html)

[Karatsuba Multiplication -- from Wolfram MathWorld](https://mathworld.wolfram.com/KaratsubaMultiplication.html)

1. Un patrón de diseño es un conjunto de técnicas utilizadas para resolver problemas comunes en el desarrollo de software. [↑](#footnote-ref-2)