Primalidad y Encriptación RSA

Alejandro Rodríguez, Rubén Palmer y Sergi Mayol

Resumen— Esta aplicación presenta una interfáz gráfica de usuario que permite identificar la primalidad y los factores de un número arbitrariamente grande. Adicionalmente, hay un "playground" donde el usuario puede encriptar, desencriptar y guardar textos mediante claves RSA de tamaño variable.

Vídeo - ver vídeo

1. Introducción

En este artículo explicaremos el funcionamiento y la implementación de nuestra práctica estructurando y dividiendo sus partes en los siguientes apartados:

Inicialmente, se citarán las librerías usadas en este proyecto. Para aquellas que se hayan desarrollado específicamente, se expondrá una breve guía de su funcionamiento y uso.

Seguidamente, se explicará cuál es la arquitectura, interfaces, funcionamiento e implementación de nuestro **M**odelo-**V**ista-**C**ontrolador (MVC) así como los métodos de mayor importancia. Para facilitar la legibilidad, se separará en cada uno de los grandes bloques de la estructura; en este caso Modelo, Vista, Controlador y Hub

Finalmente, se mostrará una breve guía de cómo usar nuestra aplicación y un ejemplo de esta.

2. Librerías

En esta sección se explicarán las librerías implementadas y usadas para llevar a cabo el desarrollo de las prácticas y permitir la reutilización de las mismas durante el transcurso de la asignatura.

El principal motivo que nos ha impulsado a implementar dichas librerías es la facilidad que proporcionan para centrarse únicamente en el desarrollo de la práctica en sí; Gracias a que durante el desarrollo de estas librerías se ha priorizado la facilidad de manejo, uso genérico y optimización para añadir el mínimo "overhead" al rendimiento del programa.

2.1. Better swing

Better swing [1] es una librería derivada de Java Swing que permite un desarrollo más amigable y sencillo de interfaces gráficas, al estar inspirado en el desarrollo web (HTML y CSS), concretamente, en el framework de "Bootstrap". Se basa en "JfreeChart" para pintar las gráficas de líneas creando "wrappers" para facilitar su uso y manejabilidad.

2.1.1. Funcionamiento

El funcionamiento del paquete es muy sencillo, básicamente, la idea es crear una o varias ventanas con una configuración deseada y acto seguido ir creando y añadiendo secciones (componentes), donde posteriormente se pueden ir borrando y actualizando los componentes individual o grupalmente. Por ello, este paquete proporciona una serie de métodos para la ventana y las secciones.

¿Cómo funciona la ventana?

Para hacer que la ventana funcione es necesario crear una instancia del objeto `Window`, inicializar la configuración de la misma empleando el método `initConfig` recibiendo por argumento la ruta donde se encuentra el

archivo. En caso contrario se empleará la predeterminada. Finalmente, para que se visualice la ventana se realizará con el método `start` que visualizará una ventana con la configuración indicada anteriormente.

En el caso de querer añadir componentes como una barra de progreso, un botón o derivados, es tan sencillo como crear una sección y emplear el método `addSection`, el cual recibe por parámetro la sección a añadir, el nombre de la sección y la posición de este en la ventana.

¿Cómo funciona la sección?

Las secciones son instancias de la clase `Section` que permiten formar los diferentes componentes de la ventana.

El funcionamiento de una sección es muy sencilla, se trata de instanciar un objeto de la clase `Section` y llamar a algún método de esta clase, pasando los parámetros adecuados, y finalmente añadir la sección a la ventana.

Otros

Adicionalmente, se dispone de una clase llamada `DirectionAndPosition`, que constituye las posibles orientaciones y direcciones que una sección puede tener.

2.1.2. Implementación

Para el desarrollo de esta librería se ha centrado en la simplicidad hacia el usuario final y la eficiencia de código mediante funciones sencillas de emplear y optimizadas para asegurar un mayor rendimiento de la interfaz de usuario. Este paquete se divide en dos principales partes:

Window: Consiste en un conjunto de funciones para la creación y configuración de la ventana.

Section: Consiste en un conjunto de funciones base para la creación de secciones en la ventana.

La implementación de la `Window` consiste en diversas partes: gestión de teclas, configuración, creación y actualización de la ventana y creación de las secciones.

La gestión de las teclas se realiza mediante una clase llamada `KeyActionManager`, que implementa la interfaz `KeyListener`. Esta clase permite gestionar los eventos de teclado de la ventana y se utiliza para la depuración y el desarrollo del programa.

En cuanto, la configuración, creación y actualización de la ventana y la creación de las secciones, se han desarrollado una serie de métodos que envuelven a un conjunto de funciones propias de java swing. Por tanto, con este conjunto de métodos y funciones se obtiene la clase `Window`.

A continuación se explicarán los principales métodos de `Better swing`:

`initConfig` es un método que permite cargar la configuración de la vista y crea el marco de la vista, incluyendo apariencia, posición, tamaño, icono, color de fondo y manejo de eventos de teclado.

`start` permite la visualización de la ventana, haciendo que la ventana sea visible para el usuario.

`stop` actua como wrapper de `JFrame::dispose`.

`addSection` permite añadir una sección a un objeto `Window` indicándole la posición y dirección del panel mediante el conjunto de variables definidas en `DirectionAndPosition`.

`updateSection` permite actualizar una sección indicándole la posición y dirección del panel mediante el conjunto de variables definidas en `DirectionAndPosition`.

`deleteComponent` permite borrar un componente específico de la ventana que se le haya pasado por parámetro al método.

`repaintComponent` permite repintar un componente específico de la ventana que se le haya pasado por parámetro al método.

`repaintAllComponents` repinta todos los componentes de la ventana.

La implementación de la `Section` consiste en diversas partes, las cuales son las siguientes: Los métodos que permiten crear las secciones ya configuradas y las clases en las que se basan los métodos anteriores.

La propia librería contiene muchas más clases y métodos que si el lector desea explorar puede acceder a su documentación a través del código fuente proporcionado en la entrega de la práctica.

2.1.3. Manual de uso

En este apartado se describe como emplear la librería para su correcto funcionamiento.

Para emplear la librería es tan sencillo como crear una instancia de la clase `Window`, llamar al método `initConfig` indicando el fichero de configuración de la ventana, si se desea, en caso contrario se deberá pasar un `null` y se emplearán la configuración por defecto, y finalmente llamar a la función `start` cuando se desee inicializar la ventana.

Es importante inicializar la configuración antes de ejecutar la función `start`, ya que en caso contrario se producirá una excepción. A continuación se muestra un sencillo ejemplo:

```
Window view = new Window();
view.initConfig("config.json");
view.start();
```

Como se observa, la librería permite cargar la configuración e iniciar la ventana independientemente, permitiendo una mayor flexibilidad.

Además, en el caso de querer reiniciar la configuración, cambiar la visibilidad de la ventana o guardar el contenido de esta sin tener que volver a compilar el código, se puede realizar a través de unos atajos de teclado, que son los siguientes:

Q: Cerrar programa.
R: Reiniciar configuración.
V: Cambiar visibilidad.
G: Guardar contenido.

Por ejemplo, al cambiar la configuración y reiniciar la ventana se aplicarán los cambios automáticamente sin compilar de nuevo el código, es decir, se permite el conocido “Hot Reloading”.

En el caso de querer crear una sección y añadirla, se realizaría de la siguiente forma:

```
Window view = new Window();
Section section = new Section();
// Los datos pueden ser de longitudes irregulares
long[][] data = { ... };
Color chartColors[] = { Color.RED, Color.BLACK };
String chartColumnLabels[] = { "Linea 1",
                               "Linea 2" };
section.createLineChart(labels,
                        data,
                        chartColors,
                        chartColumnLabels,
                        "Ejemplo Lineas");
view.addSection(section,
                DirectionAndPosition.POSITION_TOP,
                "Chart");
```

En el ejemplo anterior se crearía una gráfica de líneas con dos líneas, una de color rojo y otra negra, con los nombres “Linea 1” y “Linea 2”, con el título “Ejemplos Lineas” y con los puntos del array data.

Finalmente, comentar que hay ilimitadas posibilidades de creación y personalización de componentes con los ya configurados y las posibilidades de crear libremente los propios componentes.

3. Conceptos MVC

El Modelo Vista Controlador (MVC) es un patrón arquitectónico de software que divide una aplicación en tres elementos interconectados, separando la representación interna de la información, como se muestra al usuario y donde se hacen cómputos con esa información respectivamente.

Para esta práctica, se ha decidido modificar parcialmente este patrón al implementar la comunicación entre los módulos mediante un sistema de peticiones. Por ende, encontramos 4 módulos en nuestro MVC: Modelo, Vista, Controlador y Hub. Este último se encargará de gestionar toda la comunicación.

Esta decisión de modificación del MVC ha sido respaldada por la facilidad que obtenemos de escalar el patrón, permitiendo crear tantos módulos como queramos. Esto permitiría que, si en un futuro se quisiera añadir otro controlador en otro lenguaje de programación o encapsular controladores por funcionalidad, el único cambio a efectuar ocurriría en el hub.

A continuación se explicarán cada uno de los módulos y como han sido adaptados y empleados en la práctica.

3.1. Implementación

El Modelo vista controlador (MVC), en esta práctica, ha sido implementado como si fueran aplicaciones diferentes, es decir, cada parte del MVC es independiente de otro elemento de este ya que todos las partes se comunican a trvés de un hub. Esta implementación, trata de imitar una serie de microservicios que se comumican entre ellos a través de una API (application programming interface).

En cuanto a la implementación interna, se trata de una serie de servicios (modelo, vista y controlador) que se conectan a un servidor (hub), donde, entre ellos, se comunican a través de la red local del dispositivo.

¿Cómo hemos conseguido implementar esta idea en Java?

Para implementar el concepto de servicios y servidor se ha realizado a través de los sockets de Java, el problema, de únicamente emplear los sockets, es que los procesos se bloquean cuando envían una petición al servidor hasta que reciben una respuesta, por ello, hemos decidido crear una comunicación asíncrona no bloqueante, es decir, los servicios y el servidor no se pueden bloquear. El concepto empleado es similar al que usa el entorno en tiempo de ejecución multiplataforma NodeJs. Básicamente, un servicio realizará una petición y la respuesta que obtendrá es una promesa diciendo que le llegará una respuesta con los datos que desea, en el caso de que desee obtener información de vuelta, donde le informará de que en algún momento será notificado con los datos solicitados en la petición realizada. Con este planteamiento, se consigue que los procesos, como se ha mencionado anteriormente, no se bloquen y puedan seguir con su ejecución. En el caso, de que se desee esperar a la información, se puede realizar uso de una función similar a un "await", en la cual, se encargará de bloquear el proceso hasta que la petición tenga los datos esperados.

La siguiente imagen muestra un diagrama de cómo está implemtentado el sistema de comunicación entre los elementos:

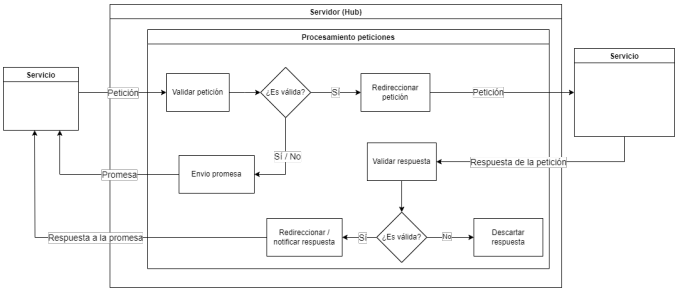

Figura 1. Comunicación entre elementos MVC y hub.

Además, de la implementación comentada anteriormente, los "endpoints"/rutas del servidor se definen a través de un archivo de configuración JSON, el cual, mapea cada petición con los servicios, al igual que sus respuestas pertinentes. Esto se puede ver en el siguiente ejemplo:

```
{
  "requestMap": [
    {
      "code": "SEND_GEOPOINTS",
      "services": ["MODEL", "CONTROLLER"]
    }
  ],
  "responseMap": [
    {
      "code": "LOAD_MAP",
      "services": ["VIEW"]
    },
  ],
}
```

3.2. Hub

El flujo de datos empleado en este patrón de diseño ha sido inspirado en cómo se comunican

los diferentes elementos en un servidor. Es decir, cualquier módulo del MVC deberá realizar una petición al servidor (Hub) y este se encargará de solventar y dar servicio a dicha petición. Se puede ver con mayor claridad cómo funciona y su implementación en la siguiente figura:

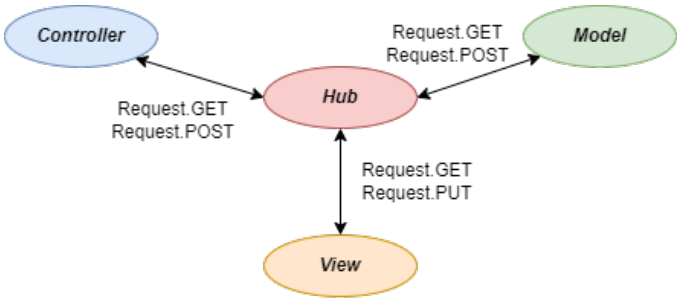

Figura 2. Patrón MVC usado en la práctica.

El Servidor/Hub se encarga de administrar los diferentes tipos de peticiones que llegan de los diferentes componentes. En la figura 2 se observan los diferentes tipos de peticiones posibles que se pueden realizar, en concreto: GET, POST y PUT. En un entorno web, GET sirve para obtener datos, POST para enviar y PUT para actualizar. En el programa los diferentes tipos de peticiones se han traducido en:

GET Realiza un retorno del dato solicitado.
POST Añade un el dato enviado.
PUT Realiza un set del dato enviado.

Para permitir esta implementación, se ha creado la interfaz `Service` con el método `notifyRequest` el cual deberá estar implementado en todos los módulos del MVC, ya que será el responsable de gestionar las peticiones y asegurar que cada uno de ellos puede realizar sus tareas designadas de manera eficiente y precisa.

3.3. Modelo

El modelo es la representación de los datos que maneja el software. Contiene los mecanismos y la lógica necesaria para acceder a la información y para actualizar el estado del modelo.

Para este proyecto, y con el objetivo de ser lo más fiel a un caso realista aplicable a una aplicación real, se ha decidido crear una `sqlite` donde se guardarán todos los datos. Así pues, nuestro modelo presenta dos elementos:

3.3.1. Base de datos

La base de datos consta del conjunto de claves RSA generadas, los puntos de la interpolación de newton para el tiempo estimado y el conjunto de resultados por cada operación. Técnicamente los atributos son:

- **Newton interpolation**: Conjunto de puntos que definen el tiempo esperado para la ejecución de obtención de factores en horas siguiendo una interpolación de newton.
- **RSA Keys**: Las claves privadas y públicas generadas históricamente durante las ejecuciones del programa.
- **Result**: Una estructura de datos que contiene un conjunto de métricas a tomar durante su ejecución. Técnicamente:
  - **result**: El objeto del cual se han obtenido los datos.
  - **time**: Tiempo de ejecución obtenido tras la obtención del dato anterior.
- **Históricos**: Un conjunto de tablas que tienen como objetivo guardar el histórico de los resultados y tiempos para la respectiva operación:
  - **Encrypt**: Encriptar un texto arbitrario.
  - **Decrypt**: Desencriptar un texto encriptado.
  - **Is Prime**: Identificar la primalidad de un número arbitrariamente grande.
  - **Get Factors**: Identificar el conjunto de factores para un número arbitrariamente grande.

3.3.2. Interfaz

Ya que el modelo es realmente la base de datos, es necesaria una interfaz que nos permita leer, modificar o escribir dentro de la base de datos. Para ello, la propia clase `Model.java` junto a `DBApi.java` nos permite ejecutar un conjunto de funciones que aplican queries a la base de datos. La mayoría de estos métodos

siguen el siguiente patrón:

- Conectar a la base de datos.
- Ejecutar el query.
- Transformar el resultado a un objeto de java (como puede ser un array).
- Devolver el resultado si no hay errores.

Aunque el diseño se acerca a la posible implementación en una aplicación real, la implicación de usar queries a una base de datos nos proporciona tanto beneficios como perjuicios.

Beneficios

- Se delega la búsqueda y filtrado de datos a un lenguaje especialmente diseñado para ello.
- Los datos más usados se guardan en la caché, mientras que los demás se pueden guardar en disco.
- Los datos se mantienen entre ejecuciones, por lo que se puede aplicar un posterior estudio de los resultados en otros entornos y lenguajes.

Perjuicios

- Se genera un overhead general en la aplicación proporcional a cómo esté programada la librería usada, ya que la conexión se debe abrir y cerrar en cada query que se ejecuta, además de la necesidad de transformar los resultados a un objeto del lenguaje usado.

Aun así, tanto por interés académico como para crear ejemplos lo más cercanos a una aplicación real, se ha decidido seguir este acercamiento ante el problema de la gestión de un modelo en una aplicación.

Finalmente, al ser un módulo de nuestro MVC, implementa la interfaz `Service` y su método `notifyRequest` que le permite recibir notificaciones de los otros módulos del MVC.

3.4. Vista

La vista contiene los componentes para representar la interfaz del usuario (IU) del programa y las herramientas con las cuales el usuario puede interactuar con los datos de la aplicación. Adicionalmente, la vista se encarga de recibir e interpretar adecuadamente los datos obtenidos del modelo. Cabe mencionar, que al igual que el resto de componentes del MVC, la clase View implementa la interfaz “Service”, la cual permite la comunicación con el resto de elementos.

`loadContent` es el encargado de cargar el contenido inicial en la ventana. Para esta práctica, carga los siguientes elementos semánticos:

`menu`, función que crea y configura el menú de utilidades de la ventana principal. En esta, se incluyen las siguientes opciones: Abrir ventanas de estadísticas, abrir el manual de usuario, cargar la base de datos, ver el ratio del mesurament y salir del programa.

`body`, función que crea, configura y actualiza la visualización de los resultados obtenidos al encriptar/desencriptar y ver la primalidad o las factores del número. Se trata de dos zonas uno para la primalidad y factores del grupo, y la segunda zona para la encriptación de texto, ya sea desde un fichero o directamente un texto.

`sidebar`, función que crea y configura la barra de opciones de la derecha de la interfaz principal. Esta permite al usuario interactuar y configurar el entorno de ejecución del algoritmo, para ello, se permite: seleccionar el número de dígitos de las claves, la semilla para generar las claves, la selección de claves generadas anteriormente y un botón para generar las claves RSA.

Adicionalmente, a la vista principal, esta permite desplegar una serie de ventanas extra. Una ventana muestra, a tiempo real, el uso y consumo de la memoria de la Java virtual machine (JVM) (3.4.1), la otra ventana muestra las estadísticas de la ejecución de los algoritmos además de su comparación (3.4.2) y un manual de usuario (4.1).

Finalmente, al ser un módulo de nuestro MVC, implementa la interfaz `Service` y su método `notifyRequest` que le permite recibir notificaciones de los otros módulos del MVC.

3.4.1. Estadísticas JVM

Este apartado de la vista principal, es el encargado de enseñar a tiempo real las estadísticas de la máquina virtual de java. Concretamente, se actualiza cada 0.5 s a partir de los datos obtenidos de la clase de java "Runtime" y muestra la memoria libre, la memoria total y el uso de esta en una gráfica de líneas, donde el eje x es instante en el tiempo que se han obtenido los datos y el eje y su valor. Todos los datos de la memoria obtenidos están en MB.

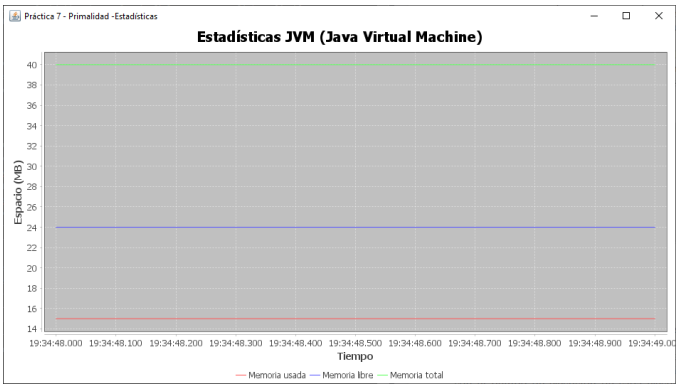

Figura 3. Interfaz estadística JVM

3.4.2. Estadísticas de los Algoritmos

Este apartado de la vista principal, es el encargado de enseñar las estadísticas de la ejecución del algoritmo de los algoritmos ejecutados. Estas estadísticas incluyen una gráfica con el tiempo de ejecución de cada ejecución del algoritmo. Los resultados (tiempos de ejecución) son representados en un gráfico de barras, donde el eje x representa el número de ejecuciones del algoritmo y el eje y el tiempo en milisegundos que ha tardado. Y finalmente, un gráfico de lineas con la evolución estimada de los tiempo de ejecución para la factorización de números primos, junto a su regreasión lineal.

Como se ha podido ver anteriormente en la imagen (4), se puede apreciar los diferentes datos obtenidos tras una serie de ejecuciones del algoritmo.

3.5. Controlador

El controlador en un MVC es el responsable de recibir y procesar la entrada del usuario y

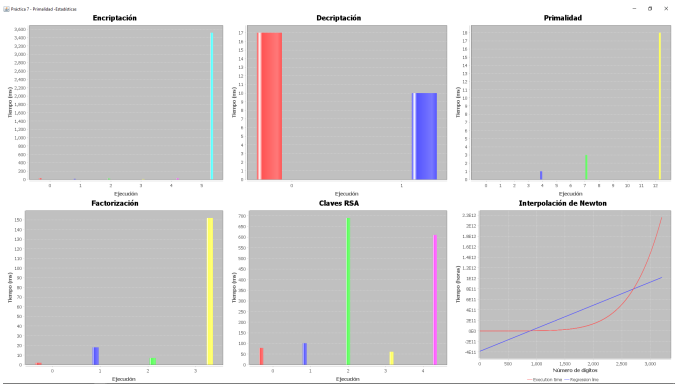

Figura 4. Interfaz estadísticas algortimos

actualizar el modelo. En esta práctica el controlador es especialmente extenso, pues es responsable de:

- Detectar si un número es primo.
- Identificar los factores de un número.
- Generar datos de estimación temporal.
- Generar claves RSA de tamaño parametrizado.
- Encriptar y desencriptar texto.
- Comprimir y descomprimir archivos.

3.5.1. Diseño

3.5.1.1. Primalidad: Para detectar si un número es primo o no, se han desarrollado un conjunto de algoritmos que el usuario, bajo su libre albedrío, puede ejecutar:

- Trial division [13]: Consiste en dividir el número que se quiere comprobar si es primo por todos los números primos menores que su raíz cuadrada. Si el número es divisible por alguno de estos números, entonces no es primo. Si no es divisible por ninguno de ellos, se considera primo.
  - **Complejidad temporal**: $O(\sqrt{N})$ ya que comprobamos en un bucle por dicha condición.
  - **Complejidad espacial**: $O(1)$ al no necesitar ninguna estructura adicional que dependa del tamaño del número.

  Adicionalmente, se ha implementado una modificación de dicho algoritmo utilizando "thread pools" [16], creando así una ejecución paralela del algoritmo.

- Fermat [14]: Basado en el teorema de fermat, establece que si un número p es primo, entonces para cualquier número entero q menor que p, se cumple que $q^{p-1} \equiv 1 \mod p$. Matemáticamente:

  $$prime(p) \longleftrightarrow \forall q \mid q < p \implies q^{p-1} \equiv 1 \mod p$$

  Para ello, se escoge un número aleatorio q y se verifica dicha congruencia. Como se puede apreciar, el algoritmo es probabilístico, lo que implica que, bajo las peores circunstancias posibles, el algoritmo podría no acabar nunca, además de poder ofrecer falsos positivos. La repetición de este proceso con diferentes valores de q aumenta la confianza de dicha congruencia. En nuestro caso, dicho número de iteraciones y semilla para la generación de valores aleatorios está parametrizada, por lo que los resultados se pueden replicar nativamente.
  - **Complejidad temporal**: Impredecible al ser un algoritmo probabilístico.
  - **Complejidad espacial**: $O(1)$ al no necesitar ninguna estructura adicional que dependa del tamaño del número.

- Miller Rabin [15]: Basado en el teorema de Miller Rabin, establece que si un número compuesto n pasa ciertas pruebas de primalidad entonces es probable que sea primo. Por cada iteración del algoritmo, es escoge un valor aleatorio q y se verifica si cumpla dicha condición. Si no la cumple se le denota como compuesto, mientras que si la cumple para todas las otras, incluida la misma, se le denota como posible primo. Al igual que en el algoritmo de Fermat, la confianza es directamente proporcional a la cantidad de iteraciones usadas.
  - **Complejidad temporal**: Impredecible al ser un algoritmo probabilístico.
  - **Complejidad espacial**: $O(1)$ al no necesitar ninguna estructura adicional que dependa del tamaño del número.

  Adicionalmente, se ha implementado una modificación de dicho algoritmo utilizando "thread pools" [16], creando así una ejecución paralela del algoritmo.

3.5.1.2. Identificación de factores: Para identificar los factores de un número arbitrariamente grande se ha desarrollado el siguiente algoritmo, cuyo pseudocódigo podría ser el siguiente:

```
def get_factors(num):
    prime_factors = {}
    divisor = 2

    while num > 1:
        if num.isPrime():
            prime_factors[num] = 1
            break

        if num % divisor != 0:
            divisor = divisor.nextPrime()
            continue

        prime_factors[divisor] =
        ↪ prime_factors.get(divisor, 0) + 1
        num = num // divisor

    return prime_factors
```

El algoritmo usa una `Map` donde sus claves vienen determinadas por los diferentes números primos que encontremos y su valor es su frecuencia; creando así un mapa de frecuencias. El caso base, definido en la línea 6, determina si el elemento ya es primo y, por ende, no existirán más factores además de él. Si no es el caso, revisamos si es divisible por el actual divisor (inicializado a 2) si no lo es, cogemos el siguiente valor primo y reiteramos el bucle. En el caso de que si sea divisible, hemos encontrado un factor, por lo que lo añadimos al mapa de frecuencias y modificamos el valor de nuestro número. Al final, después de todo este proceso, retornamos dicho mapa.

La complejidad temporal de dicho algoritmo depende de varios factores:

- Implementación de los métodos `isPrime` y `nextPrime` que, al estar dentro de un bucle, podrían afectar gravemente al rendimiento de la aplicación.
- La búsqueda de factores primos como tal que, en el peor de los casos y para un número no primo, se ejecutará hasta que dicho se reduzca a 1. En general, podemos asumir que el número de iteraciones depende de la cantidad de factores primos que tenga dicho número y por ende será proporcional al dicho.

Así pues, la complejidad temporal asintótica de este método se podría expresar como $O(K * F)$ donde K es el coste de implementación de los previos métodos y F la cantidad de factores que tiene un número.

Espacialmente, al estar generando un mapa de frecuencias, se toma como $O(F)$ donde F es la cantidad de factores para un número arbitrario, al tener que crear una entrada en el mapa por cada factor diferente.

3.5.1.3. Generación de datos temporales: Para evitar esperas significativamente altas a la hora de identificar los factores de un número arbitrario, se ha implementado un breve estudio que permite estimar su tiempo de ejecución. Comentar que, debido al uso de una interpolación polinómica de newton dicha estimación depende del hardware en el que se ejecute el muestreo de datos y, por ende, no será preciso en otras máquinas. Para ello se usó planetcalc que permite, identificando un conjunto de puntos, efectuar dicha interpolación. Así pues podemos obtener el siguiente resultado:

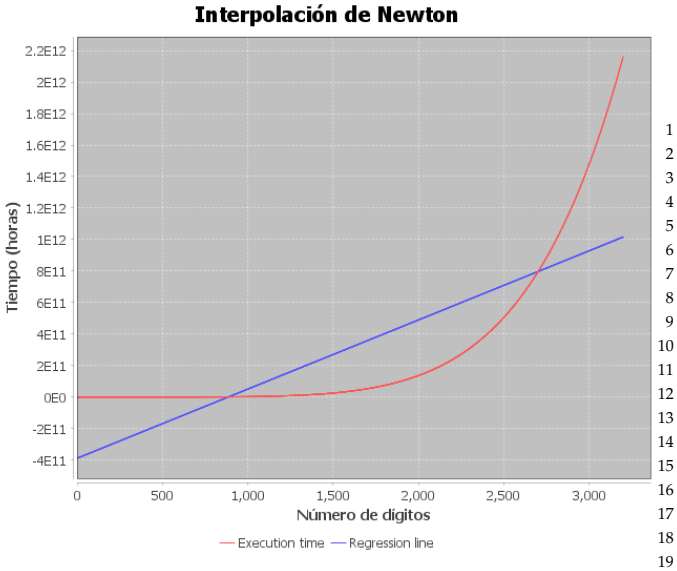

Figura 5. Interpolación de newton

Como se puede apreciar, podemos ver la propia función en rojo y la regresión lineal en azul. Con respecto a su forma, aun sabiendo que es una polinómica de sexto grado, se podría decir que, bajo el dominio $[0, \infty)$ presenta una curva exponencial. Así pues, la función que devuelve los factores de un número dado primero evalúa si, con respecto a esta función, dicha ejecución tardaría más que cierto límite (en nuestro caso, un minuto); Si es así, no se calcula y se le informa al usuario cuanto tiempo aproximadamente tardaría dicho cálculo.

3.5.1.4. Generación de claves RSA: Para dicha generación se han implementado dos métodos. El primero genera un número primo de X cifras, usado para poder determinar *aproximadamente* el tamaño de la clave RSA. Esto se debe a que, para crear dicha clave, se generaran dos números de longitud X que se usarán como los parámetros p y q. Ya que los multiplicamos para determinar n, y bajo aproximación, determinamos que la longitud de un número A sigue el siguiente teorema:

$$\forall A, A = B \cdot C \implies \mid A \mid = \mid B \mid + \mid C \mid$$

Así pues, y mediante una implementación probabilística, generamos números de X cantidad de valores e identificamos si es primo; en tal caso lo devolvemos.

El segundo algoritmo genera la propia pareja de claves RSA, cuyo pseudocódigo, podría ser el siguiente:

```
def generate_key_pair(p, q, seed):
    n = p * q

    phi = (p - 1) * (q - 1)

    rng = random(seed)

    e = None
    while True:
        e = rng.randint(1, phi - 1)
        if gcd(e, phi) == 1:
            break

    d = powmod(e, -1, phi)

    return KeyPair(
        PrivateKey(d, n),
        PublicKey(e, n)
    )
```

Donde se efectúan los siguientes pasos:

1. Se genera n como el producto de p y q.
2. Se obtiene $\phi(n)$ como $(p-1) \cdot (q-1)$.
3. Se calcula e tal que $0 < e < \phi(n)$ y tanto e como $\phi(n)$ son coprimos mediante un bucle.

4. Se calcula d como el inverso multiplicativo modular [18] de $e \mod \phi(n)$.
5. Finalmente, se retorna la pareja de claves RSA

3.5.1.5. Encriptación: Una pareja de claves RSA contiene un conjunto de métodos que permiten encriptar y desencriptar tanto texto como ficheros. Ambos utilizan un metodo de exponenciación modular [17] cuya implementación sigue el siguiente pseudocódigo:

```
def modular_exponentiation(base, exponent,
↪  modulus):
    result = 1

    while exponent > 0:
        if exponent & 1:
            result = (result * base) % modulus
        base = (base * base) % modulus
        exponent >>= 1

    return result
```

Esencialmente, solo podemos encriptar (y por ende desencriptar) números; por ello, en el caso de querer encriptar un texto, debemos iterar por todos sus caracteres, convertirlos en sus valores ASCII y encriptar dicho valor; lo que se podría implementar de la siguiente manera:

```
for (String string : text.split("\n")) {
    string.chars().forEach(e ->
        encryptedText
            .append(
                publicKey
                    .encrypt(String.valueOf(e))
                    .toString()
            )
            .append("\n");
        );
        // Add special character to indicate EOL
        encryptedText.append("@\n");
}
```

Como se puede apreciar, iteramos por todas las líneas y por cada valor ASCII los encriptamos y añadimos al string encriptado. Adicionalmente, se puede ver como añadimos un terminal "@" al final de cada línea tratada. Esto nos permite fácilmente reconvertir el texto completo, ya que transformar el final de la línea en Linux y Windows nos daba ligeros problemas de inconsistencia.

El proceso de des encriptación sigue la misma estructura, desencriptando cada línea a un carácter y transformando el terminal en un salto de línea, como se puede ver aquí:

```
for (String string : text.split("\n")) {
        if (string.equals("@")) {
                decryptedText.append("\n");
                continue;
        }

        decryptedText
        .append((char) privateKey
            .decrypt(string)
            .intValue()
        );
}
```

3.5.1.6. Compresión: Para este proyecto se ha añadido la posibilidad de guardar un archivo encriptado o desencriptado a disco. Adicionalmente, y para reducir el tamaño de dichos archivos, se ha decidido comprimirlos mediante el formato de compresión ZIP, usando `Deflater` y `Inflater` que permiten, respectivamente, comprimir y descomprimir "streams" de datos. Esto nos permite, de media, reducir hasta cinco veces el tamaño de los archivos guardados. Además de ello, la aplicación puede leer archivos tanto encriptados como no encriptados, debido al uso de una extensión especifica que se fuerza a la hora de guardar un archivo desde la aplicación.

3.5.2. Estudios

3.5.2.1. Detección números primos: Para el estudio del rendimiento de los algoritmos, se ha realizado una serie de tests. Estos tests consisten en la ejecución de los algoritmos, concretamente, 10000 veces para un determinado número primo de 1683 dígitos. A partir de los datos obtenidos se han guardado en una base de datos para poder analizar los datos con "Jupyter Notebooks" para obtener una mayor facilidad y versatilidad del análisis de los datos. En estos "notebooks", se analizan profundamente los resultados obtenidos de los tests, específicamente, se analiza la estabilidad y el rendimiento en tiempo de ejecución de estos algoritmos.

A partir del análisis realizado, se obtiene la conclusión de que los datos obtenidos son similares entre ellos, aun así se pueden

diferenciar con claridad entre ellos. Además, se puede observar cuanto de estables con los algoritmos, el más estable es el que tiene menos anomalías en los datos, es decir, el que tiene menos picos y valles, en este caso, el "millerRabin" en paralelo y el menos estable, en este, el "miller Rabin" iterativo. Aun así, que sea más estable no significa que sea más eficiente, ya que puede ser más estable pero tardar más en ejecutarse y viceversa, ya que, el que tiene de media un menor tiempo de ejecución es la división iterativa en paralelo del número primo y el más lento de media la división iterativa del número primo.

Por lo que, a la hora de elegir un algoritmo se debe tener en cuenta tanto su estabilidad como su eficiencia.

3.5.2.2. Factorización números: Para el estudio del rendimiento de los algoritmos, se ha empleado una estimación de lo que se tardaría en obtener los factores del número, ya que la factorización de un número es muy costosa. Concretamente, se ha empleado la interpolación de newton, donde la entrada es el número de cifras y la salida el tiempo estimado en horas. A partir de los datos obtenidos se han guardado en una base de datos para poder analizar los datos con "Jupyter Notebooks" para obtener una mayor facilidad y versatilidad del análisis de los datos, al igual que en el anterior estudio.

En este estudio, únicamente, se visualiza la estimación del coste junto a su regresión lineal para apreciar el coste de factorizar un número.

Para ver con más detalle como se han analizado los datos y los resultados, de los apartados 3.5.2.1 y 3.5.2.2, se pueden encontrar en los PDF adjuntos y en la carpeta "tools". En el directorio "tools" contiene el código fuente de los tests y los "Jupyter notebooks".

4. MANUAL DE USUARIO

Para el correcto uso de la aplicación, el conjunto de acciones que se pueden realizar en dicho programa serán definidas a continuación. También, la distribución de las secciones de la interfaz junto a su explicación y funcionalidad.

Cuando se ejecute el programa por primera vez en la pantalla se debe mostrar la siguiente interfaz de usuario:

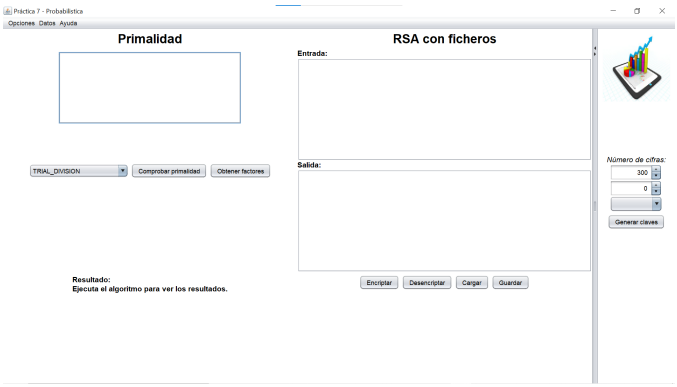

Figura 6. Interfaz de usuario.

4.1. Menu

El menú de la aplicación consiste en la barra que se sitúa debajo del marco superior de la ventana. En esta se puede encontrar un conjunto de opciones para que el usuario pueda interactuar con la aplicación, modificar y analizar el comportamiento de esta. En concreto, se encuentran las opciones de "Opciones", "Datos" y "Ayuda", respectivamente.

En la primera se sitúan las acciones para salir del programa y para iniciar las ventanas de estadísticas explicadas anteriormente en los apartados 3.4.1 y 3.4.2.

En la segunda opción llamada "Datos" encontramos en primer lugar, la opción para crear una base de datos y cargar los datos. Por otra parte, tenemos la opción de obtener el resultado de la librería "Mesurament".

En la tercera opción se encuentra un menú desplegable con un manual de usuario con la explicación del funcionamiento de la aplicación.

4.2. Main

El “Main” es el bloque principal de la vista, donde el usuario podrá interactuar con múltiples opciones y apartados que se explicarán a continuación:

En la parte izquierda tenemos el panel de “Primalidad”, aquí el usuario podrá introducir un número y comprobar si es primo con el botón de “Comprobar primalidad”. Bajo a éste se encuentra el botón “Obtener factores”, con el que se calculará la factorización del número introducido en el panel superior. Por otra parte, también se han añadido diferentes métodos de cálculo de primalidad en un selector, se han implementado los siguientes métodos:

- Trial Division
- Fermat
- Miller Rabin
- Miller Rabin Parallel
- Trial Division Parallel

El resultado será mostrado por pantalla, además del tiempo que ha tardado en ser calculado. Si no se introduce un número, la aplicación avisará al usuario de que no es posible.

En la parte derecha tenemos en panel de “RSA con ficheros”, con dos pequeños paneles, el superior “Entrada”, donde el usuario podrá introducir el texto. El panel inferior “Salida” será donde se generará el texto encriptado o desencriptado. Bajo a éste se encuentran los botones “Encriptar” y “Desencriptar” respectivamente. Además, se han añadido los siguientes botones, “Cargar” para cargar textos directamente y el botón “Guardar” para guardar la salida del texto.

Cabe destacar que para encriptar o desencriptar el texto se deben usar las claves.

4.3. Sidebar

El “Sidebar” contiene un conjunto de opciones para modificar los datos y el modo de ejecución de la aplicación. A continuación, se explicará cada una de estas opciones y su función.

En primer lugar, se encuentra la opción para seleccionar el “Número de cifras”. En esta, se puede escoger el número de cifras para generar las claves.

En segundo lugar, se encuentra un selector de las claves generadas y guardadas en base de datos. Tras clicar el botón “Cargar BD” del menú, se podrá seleccionar una key del selector.

4.4. Ejemplo ejecución

Al iniciar la aplicación se mostrará una interfaz como la expuesta en la imagen 6. Por una parte, el usuario podrá introducir un número y calcular si es primo o sus factores. En el otro lugar, se podrá introducir un texto, y tras generar unas claves encriptar o desencriptar el texto. A continuación, la siguiente imagen, muestra un ejemplo de la interfaz tras la ejecución:

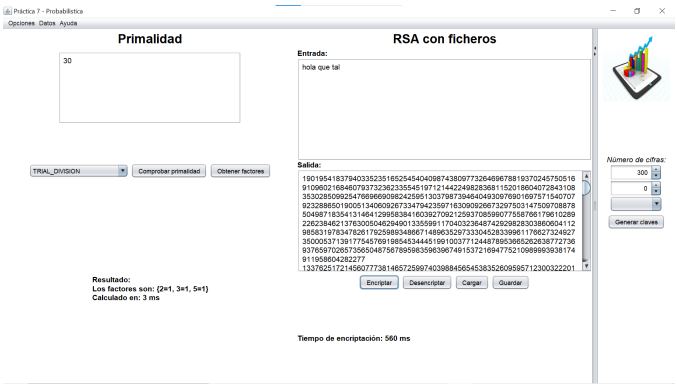

Figura 7. Interfaz de usuario en ejecución.

A partir de aquí, el usuario puede salir del programa con el botón del menú, cargar una base de datos, mostrar el resultado de mesurament, calcular la primalidad o los factores de otro número, cambiar el método de primalidad, cambiar el número de cifras, encriptar o desencriptar texto y ver estadísticas de la ejecución, ver apartados 3.4.1 y 3.4.2.

5. Conclusión

Teniendo en cuenta el desarrollo expuesto, la aplicación permite generar un punto de referencia de un conjunto de algoritmos de diferente complejidad asintótica mediante una representación gráfica e interactiva de la media

de los tiempos de ejecución a través de unos parámetros de entrada definidos por el usuario. Adicionalmente, se ha implementado mediante una versión modificada del patrón de diseño de software **M**odelo **V**ista **C**ontrolador (MVC) añadiendo un cuatro módulos que permite la comunicación entre los diferentes elementos de este mediante peticiones. Esta modificación ha sido fuertemente inspirada en el diseño de un servidor, lo que permite gran flexibilidad y escalabilidad de desarrollo, al poder interceptar y gestionar las peticiones a gusto del desarrollador.

Además, se ha hecho énfasis en el uso de las librerías creadas por los miembros del grupo para facilitar el trabajo y la reutilización de código en futuras prácticas. Reiterar que el principal motivo que nos ha impulsado a implementar dichas librerías es la facilidad que proporcionan para centrarse únicamente en el desarrollo de la práctica en sí; Gracias a que durante el desarrollo de estas librerías se ha priorizado la facilidad de manejo, uso genérico y optimización para añadir el mínimo "overhead" al rendimiento del programa.

Concluir que, con respecto al ámbito académico, este proyecto nos ha permitido consolidar el concepto de patrón de diseño MVC gracias a un primer proceso de investigación y discusión del diseño a implementar entre los integrantes del grupo.

6. Distribución del trabajo realizado

El trabajo realizado por cada miembro de la práctica ha sido el siguiente:

- Alejandro Rodríguez:
  - Documentación
  - Desarrollador de la UI del proyecto
  - Desarrollador de la guía de usuario
  - Desarrollador de la plantilla base del proyecto
- Rubén Palmer:
  - Documentación
  - Diseño del proyecto
  - Principal desarrollador del "backend" de la aplicación
  - Desarrollador general de la aplicación
  - Desarrollador del "frontend"
  - Diseñador de la implementación del patrón MVC de la aplicación
  - Desarrollador de la plantilla base del proyecto
  - Desarrollador de la plantilla de la documentación
- Sergi Mayol:
  - Documentación
  - Diseño del proyecto
  - Desarrollador de la librería `Better Swing`
  - Principal desarrollador del "frontend" de la aplicación
  - Desarrollador general de la aplicación
  - Desarrollador del "backend" de la aplicación
  - Diseñador de la implementación del patrón MVC de la aplicación
  - Desarrollador de la plantilla base del proyecto

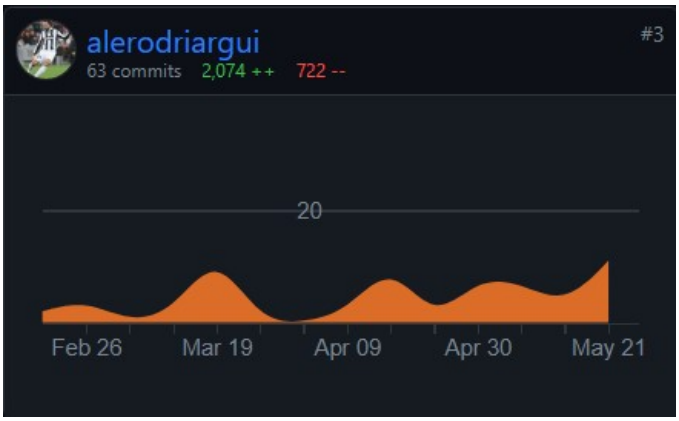

Figura 8. Contribución en el proyecto de Alejandro Rodríguez

Reconocimientos

- Se agradece la colaboración del Dr. Miquel Mascaró Portells en la resolución de dudas y supervisión del proyecto.

Referencias

[1] Better Swing, *An easy way to develop java GUI apps*, V0.0.3. Ver documentación completa aquí.

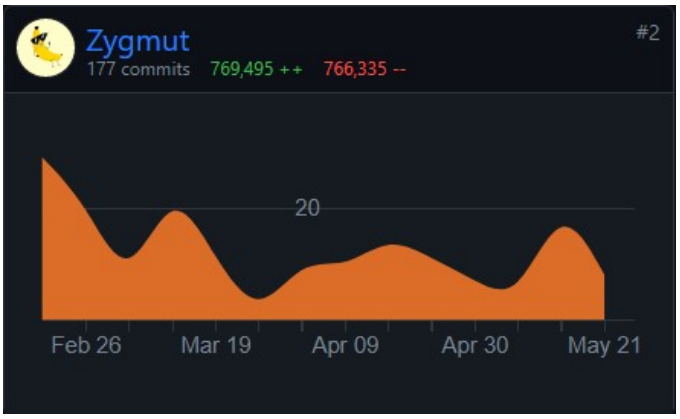

Figura 9. Contribución en el proyecto de Rubén Palmer

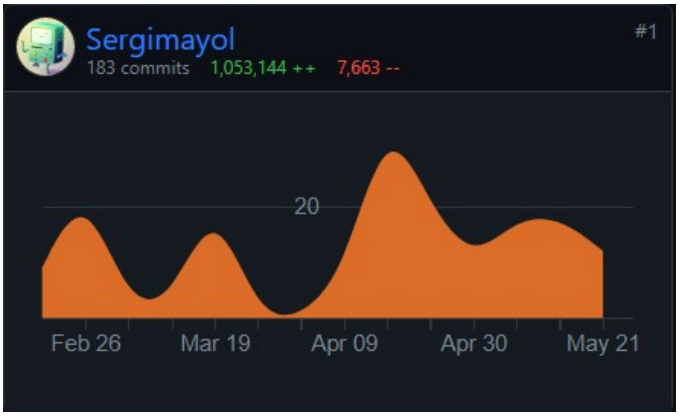

Figura 10. Contribución en el proyecto de Sergi Mayol

[2] Graphical engine, *How to develop a graphical engine from scratch*. Ver enlace aquí.
[3] Google json (Gson), *Java serialization/deserialization library to convert Java Objects into JSON and back*. Ver enlace aquí.
[4] Node.js Architecture, *Architecture of a single thread cross-platform, open-source server environment*. Ver enlace aquí.
[5] The SQLite database, a serverless database. *Quick start and documentation*. Ver enlace aquí.
[6] Open source Java chart library. *An easy way to create and visualize charts in Java*. Ver enlace aquí.
[7] A way to define different behavior for a method with the same name. Ver enlace aquí.
[8] What is RSA? How does it work?. Ver enlace aquí.
[9] How to check if a number is prime. Ver enlace aquí.
[10] Java biginteger implementation and api docs reference. Ver enlace aquí.
[11] Pandas and numpy for data analysis and visualization. Ver enlace Pandas: aquí. Ver enlace Numpy: aquí .
[12] Jupyter notebooks for data analysis and visualization. Ver enlace aquí.
[13] Trial division for integer factorization. Ver enlace aquí.
[14] Fermat primality test to determine whether a number is a probable prime. Ver enlace aquí.
[15] Miller–Rabin primality test to determine whether a number is a probable prime. Ver enlace aquí.
[16] How to work with thread pools in Java. Ver enlace aquí.
[17] Modular exponentiation. Ver enlace aquí.
[18] Modular multiplicative inverse. Ver enlace aquí.