

## 1 Detalles de implementación

### 1.1 Modo 1

Para lograr generar referencias de página en primer lugar se define una posición inicial en páginas y desplazamiento para cada una de las 3 matrices dependiendo del tamaño del entero, el tamaño de una página y el tamaño de la matriz. Una vez guardadas en distintas variables la página y desplazamiento inicial de cada matriz se inicia un ciclo para calcular la posición de cada entero de cada matriz. Para lograr esto se itera con un doble ciclo cada posición de las matrices (pues tienen mismo número de filas y columnas de modo que las posiciones coinciden). Para cada posición y para cada una de las 3 matrices se guarda en un String (que posteriormente se transformara en un archivo) la página y desplazamiento actual del entero de la posición actual de cada matriz. Es decir que por cada ciclo se almacena la posición equivalente de las tres matrices (pues así se calcula la suma). Posteriormente se actualiza la página y desplazamiento actual de cada matriz usando divisiones enteras y residuos. Para actualizar la página se le suma a la variable “el valor de sumar el desplazamiento actual y el tamaño de entero, dividido en una división entera por el tamaño de una página”. De forma que si la suma del desplazamiento actual y el tamaño de entero es menor al tamaño de página, la página permanece igual; y si es mayor, se aumenta en una posición la página. Respecto al desplazamiento se realiza la misma operación, pero en lugar de sumar a la variable se realiza una asignación y en lugar de hacer división entera se realiza la operación con módulo. De forma que si la suma del desplazamiento actual y el tamaño de entero es menor al tamaño de página, el desplazamiento se le suma el tamaño de entero; y si es mayor, se vuelve a iniciar en 0. Finalmente se transforma esto a un archivo.

### 1.2 Modo 2

#### 1.2.1 Algoritmo de Envejecimiento.

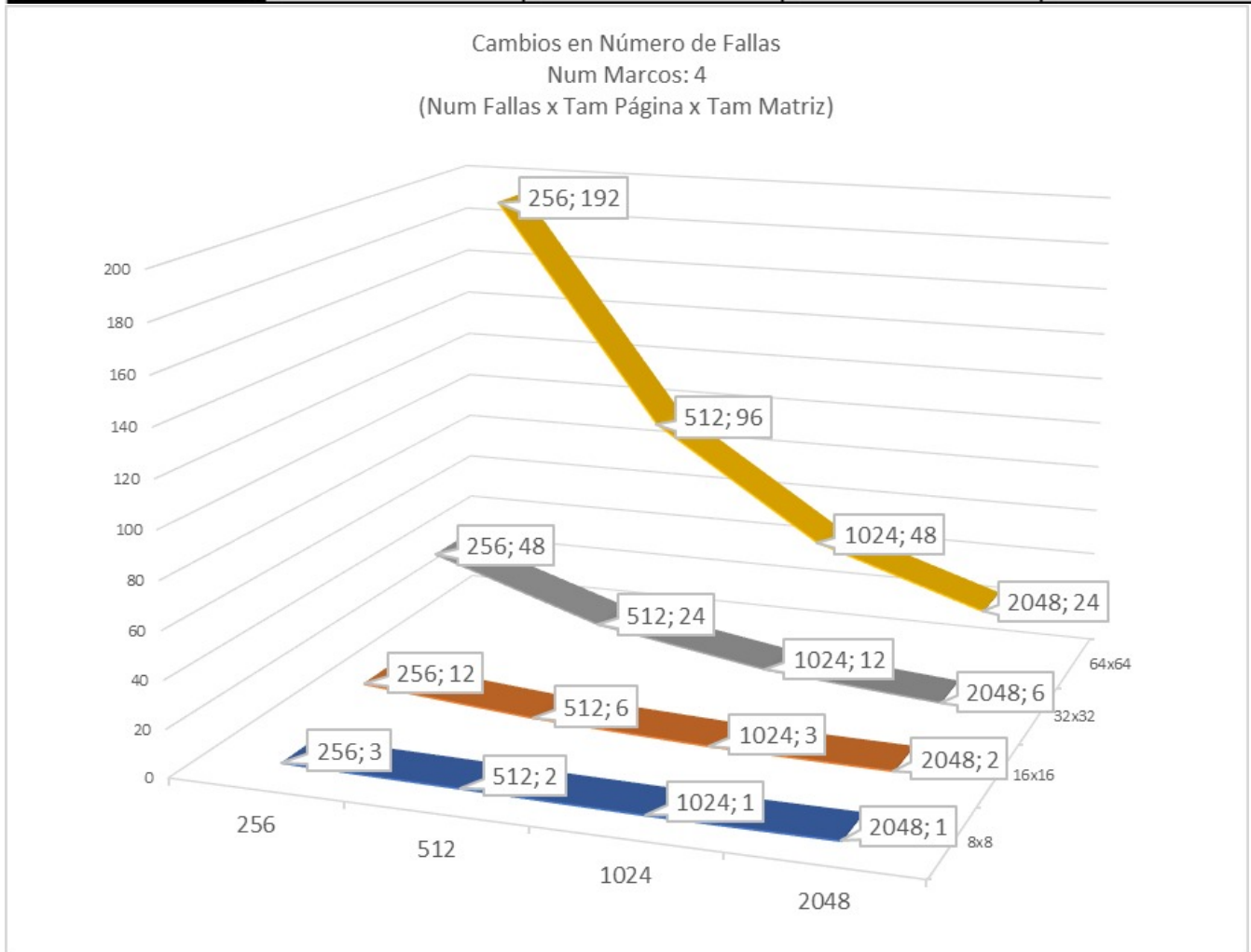
Con respecto al algoritmo de envejecimiento, este se encuentra localizado en el Manejador, principalmente con el fin de poder acceder a la tabla de páginas de forma relativamente sencilla, cambiando valores como la edad binaria de la página o el booleano que indica en memoria principal si se encuentra o no. Esto reduce el tiempo de ejecución de la operación, lo cuál es importante cuando consideramos que esta es una operación realizada varias veces por segundo.

Las condiciones y parámetros de las páginas están definidos en la clase 'Página', por lo que los 2 algoritmos principales del proceso de envejecimiento pueden consultar una misma estructura y hacer cambios acordemente. El primer algoritmo itera sobre la tabla de páginas, haciendo operaciones binarias sobre el int que representa la edad de la página, mientras que el otro selecciona la página de menor edad.

## 2 Gráficos de ejecución

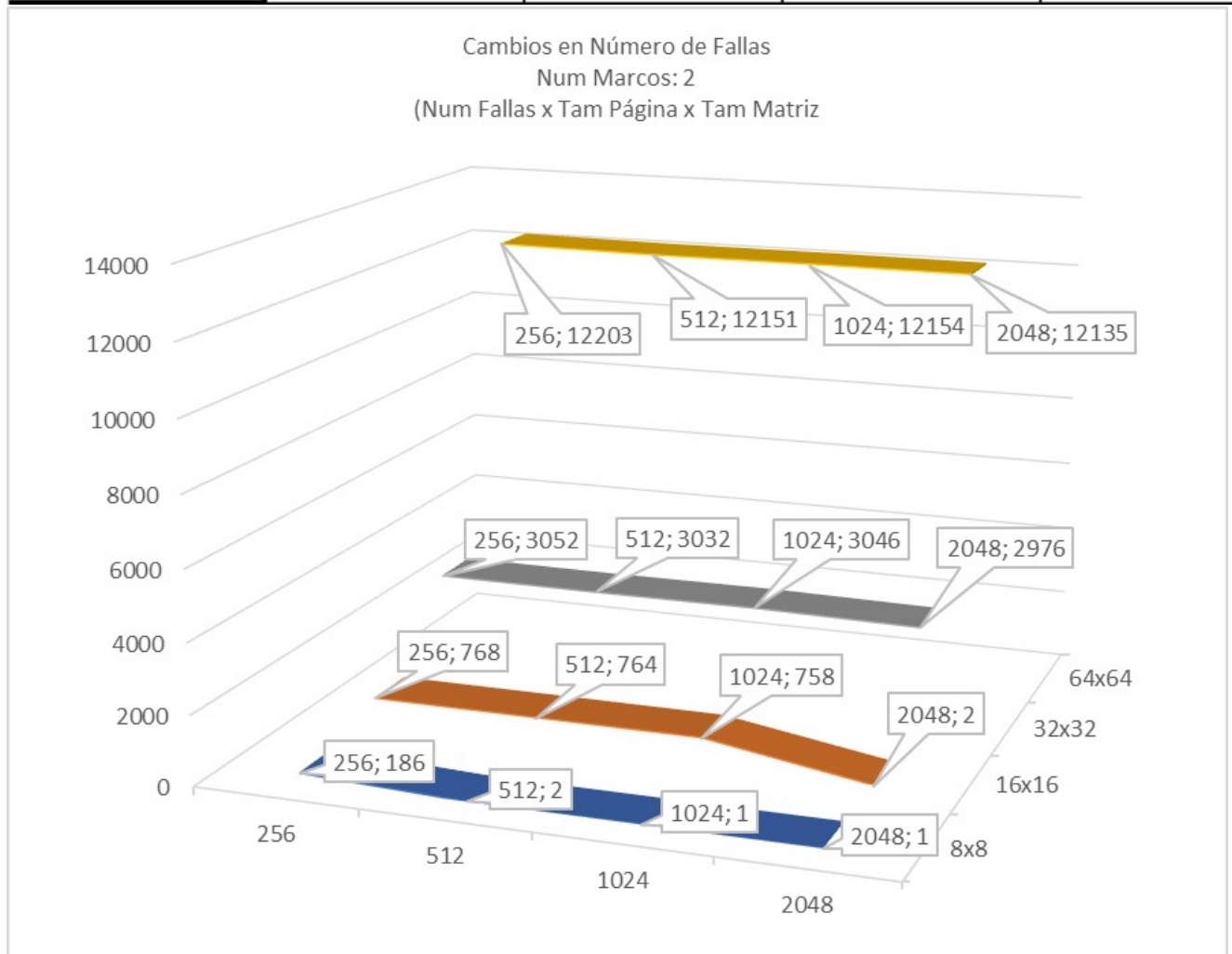
### 2.1 Prueba 1

	Tam Pagina			
Tam Matriz	256	512	1024	2048
8x8	3	2	1	1
16x16	12	6	3	2
32x32	48	24	12	6
64x64	192	96	48	24



## 2.2 Prueba 2

	Tam Pagina			
Tam Matriz	256	512	1024	2048
8x8	186	2	1	1
16x16	768	764	758	2
32x32	3052	3032	3046	2976
64x64	12203	12151	12154	12135



## 3 Preguntas realizadas.

### 3.1 ¿Qué estructuras de datos se utilizaron y porque? (cuándo se actualizan, con base en qué y en qué consiste la actualización).

En general, esta solución al problema no requiere ninguna dependencia que no esté en la base de Java 1.8. Themurin y, al leer esta solución, es bastante evidente que fue en realidad una solución bastante orientada a trabajar con las fortalezas de Java. Entonces pues, encontremos las clases propuestas y con ellas, sus atributos. Si fuésemos a considerar una clase como una estructura de datos de alto nivel, bajo la lógica del lenguaje usado, nos daríamos cuenta de que la clase 'Pagina.java' es en realidad, bastante parecida a como se utilizaría un struct en C, y que su principal función es almacenar y actualizar información de una página en concreto. Esta luego se almacenará en arrays de Páginas, que representarán las estructuras del sistema operativo que almacenan estas páginas de memoria. Estos arrays corresponden a MemoriaVirtual, MarcoPagina y TablaDePaginas, cuyos

nombres son indicativos de las definiciones a estos términos establecidas de forma teórica, con MemoriaVirtual apuntando a todas las tablas contenidas en la memoria virtual, MarcoPagina indicando las páginas en memoria real, y TablaDePaginas siendo un array complementario a este último, que almacena la información de índices dentro de la memoria real. Iterando sobre estos arrays podemos consultar fácilmente las páginas dentro de ciclos y acceder a estas durante la ejecución de los algoritmos. Esto es particularmente evidente en el modo 2, donde el algoritmo de envejecimiento y de actualización constantemente deben acceder y modificar páginas, en caso de ser de interés para un algoritmo en concreto, este accederá a la información contenida y la cambiará, como los algoritmos están sincronizados podemos garantizar hasta cierto punto la consistencia interna de la información y poder mantener registro de todo, fuera de esto se llevó cuenta de valores numéricos importantes con int y se usaron Scanners para poder leer y escribir archivos en el programa. El manejo de estas estructuras puede en algunos casos llegar a tener en cuenta el orden de las operaciones, como ocurre con MarcoPagina, por ejemplo.

### **3.2 ¿Que esquema de sincronización fue usado? Justifique brevemente dónde es necesario usar sincronización y por qué**

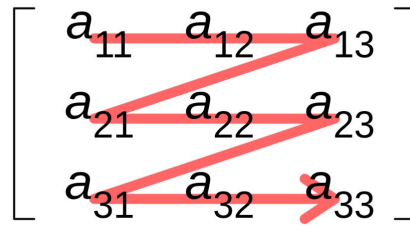
Para lograr la sincronización entre los threads del algoritmo de reemplazo de páginas y el de envejecimiento se usa una clase monitor que se instancia una única vez para almacenar las estructuras compartidas entre ambos threads. Dentro de esta instancia única de monitor se tienen métodos sincronizados (synchronized) sobre los get de varias variables o sobre la variable “continuar” que permanecerá en verdadero hasta que termine el thread de reemplazo de páginas para avisar al de envejecimiento que debe terminar. O también sobre el método de almacenar memoria virtual donde se crean las páginas con los valores por defecto, esto ya que si los otros hilos comienzan a ejecutarse antes de instanciar todas las páginas podrían haber lecturas sucias del tamaño de la memoria virtual. También sobre el método agregar página ya que se encarga de forma atómica de reemplazar páginas en marcos de página cuando haya un fallo de página (eliminación y agregación), además de que consulta el resultado del algoritmo de envejecimiento. Y, finalmente, sobre el método de envejecimiento ya que este no puede correr a la vez que el otro thread o si no podría haber una página usada recientemente que no sea tomada en cuenta por el algoritmo de envejecimiento.

### **3.3 Esquema de sincronización usado. Justifique brevemente dónde es necesario usar sincronización y por qué**

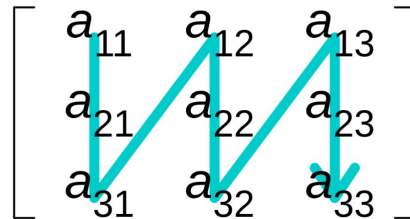
### **3.4 ¿Que pasaría si las matrices fueran almacenadas en major column order?**

En caso de que pudiésemos cambiar el orden de ejecución de la suma no habría mayor diferencia, pues sería muy similar al caso actual. Sin embargo, si la suma se siguiera haciendo en orden de las filas pero las matrices fueran almacenadas siguiendo column-major order ya se empezaría a notar la diferencia. Ya que analizando como funciona el algoritmo actualmente, podemos ver que cuando en los marcos de página caben al menos tres matrices a la vez, no se generan tantos fallos al leer un gran número de elementos de una misma página en una sola subida a memoria física. Es decir que es probable que los elementos ubicados en una misma fila estén en la misma página y no haya tantos fallos de página. Mientras que en un column-major order la probabilidad de que dos elementos de una misma fila se encuentren en una misma página disminuiría bastante. Por lo que es posible que para cada elemento de la matriz no se encuentre la página necesaria en los marcos de página, aumentando considerablemente los fallos de página.

### Row-major order



### Column-major order



### 3.5 ¿Cómo varía el número de fallas de página si el algoritmo estudiado es el de multiplicación de matrices?

La suma es una operación cuyo resultado generalmente no excede el tamaño de los sumandos, es decir, no requiere de una mayor cantidad de bits para ser representado en su forma binaria. Teniendo esto en cuenta y considerando que en el algoritmo solo se estudiaron tamaños de entero inferiores al tamaño de una página, se puede afirmar que para cargar los tres números de la suma (cada uno proveniente de su respectiva matriz) se requiere de máximo tres marcos de página. Por otro lado, dada la naturaleza de la multiplicación, es muchísimo más probable que el resultado adquiera un valor superior al de los factores, de manera que, en las mismas condiciones de la suma, será necesaria más de una página para almacenar este valor en su totalidad. Esto ya que para multiplicar matrices se tendría que realizar un producto punto para cada posición de la matriz resultado (recordar la fórmula de producto punto):

$$\vec{u} \cdot \vec{v} = u_1 \cdot v_1 + u_2 \cdot v_2$$

Y, además, este cálculo se realiza sobre la fila de una matriz y la columna de la otra, de forma que sería probable que se deban usar más páginas para guardar los cálculos parciales. Habiendo comprendido como se desenvuelven ambos algoritmos al cargarse en memoria virtual, llega el momento de revisar qué sucede cuando se quieren cargar a memoria real, lugar donde suceden las fallas de página: como ya se ha comprobado que en promedio una operación de multiplicación requiere de más páginas que una de suma, también necesitará de más marcos de página para cargarse en memoria real. Como resultado, se producirá una mayor cantidad de fallas al ser necesario reemplazar más frecuentemente las páginas para poder cargar los números que se requieran.