Juan José Beltrán Ruiz – 201819446

Santiago Bobadilla Suárez – 201820728

# Caso de Estudio 2 – Memoria Virtual

Contenido

[Caso de Estudio 2 – Memoria Virtual 1](#_Toc68989014)

[Descripción de las estructuras de datos usadas para simular el comportamiento del sistema de paginación y cómo usa dichas estructuras (cuándo se actualizan, con base en qué y en qué consiste la actualización). 2](#_Toc68989015)

[Esquema de sincronización usado. Justifique brevemente dónde es necesario usar sincronización y por qué. 4](#_Toc68989016)

[Una tabla con los datos recopilados (número de fallas de página por cada caso simulado). 7](#_Toc68989017)

[Una serie de gráficas que ilustren el comportamiento del sistema. Para eso cree una gráfica por cada tamaño de programa estudiado y en cada una ilustre: número de fallas de página vs. número de marcos asignados vs. nivel de localidad. 7](#_Toc68989018)

[Número de Páginas del Proceso 9](#_Toc68989019)

[Nivel de Localidad 10](#_Toc68989020)

[Escriba su interpretación de los resultados: ¿tienen sentido? Justifique su respuesta. 11](#_Toc68989021)

[Número de Marcos Asignados 11](#_Toc68989022)

[Número de Páginas del Proceso 11](#_Toc68989023)

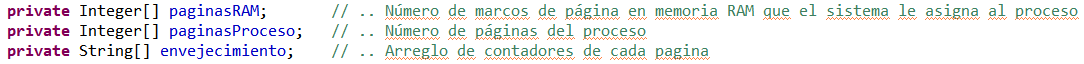
[Nivel de Localidad 11](#_Toc68989024)

## Descripción de las estructuras de datos usadas para simular el comportamiento del sistema de paginación y cómo usa dichas estructuras (cuándo se actualizan, con base en qué y en qué consiste la actualización).

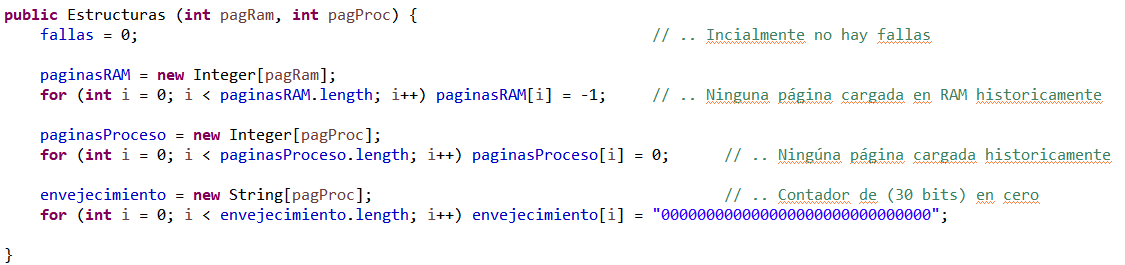
En primera instancia, se debe aclarar que el objetivo de las estructuras usadas y los métodos implementados fue obtener el número de fallas de una secuencia de páginas (con ciertas propiedades en particular) por medio del algoritmo de envejecimiento.

Este algoritmo es una modificación del LRU, donde por medio de 1 y 0 se lleva un registro histórico significativo de las páginas usadas recientemente. Este registro se expresa como un contador independiente para cada página el cual maneja 30 bits, y es significativo porque el mayor registro está a la izquierda. Es decir, se mueven los datos a la derecha y aquella página que se acaba de cargar se le pone 1 (a la izquierda) y al resto de páginas un 0. Así el menor contador pertenece a aquella página menos usada.

En cuanto a las estructuras y métodos (con el fin de ejemplificar de mejor manera el algoritmo) se hizo uso de tres arreglos. En el primero ‘paginasRAM’ hace referencia de las páginas en memoria RAM que se tienen, o de otra forma, que ya están cargadas en ella. Es decir, cuando una página vaya a ser cargada se hará respectivamente en este arreglo. La segunda ‘paginasProceso’ es una estructura intermedia donde se registra la última página cargada con el fin de facilitar la actualización de los contadores, teniendo así los identificadores de las páginas dentro del proceso y un 1 en caso de que la página esté cargada en un momento determinado y un 0 de lo contrario. Tercera y última, ‘envejecimiento’, lleva el registro de los contadores de cada página. Cabe resaltar que, si bien las dos primeras guardan el número de la página, el contador se manejara como un String de 30 caracteres (alusión a los 30 bits).



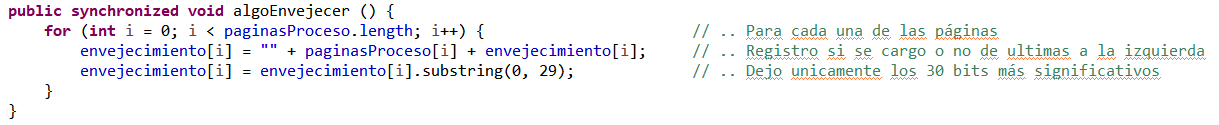
Al momento de inicializar las estructuras y con el fin de evitar errores o elementos nulos cada arreglo se inicializó de manera lógica. Las ‘paginasRAM’ empiezan en -1, alusión a que no hay nada cargado. Las ‘paginasProceso’ empiezan en 0, mostrando que ninguna se ha cargado; y ‘envejecimiento’ empieza con una cadena de treinta ceros que muestra un registro del contador vacío para todas las páginas.



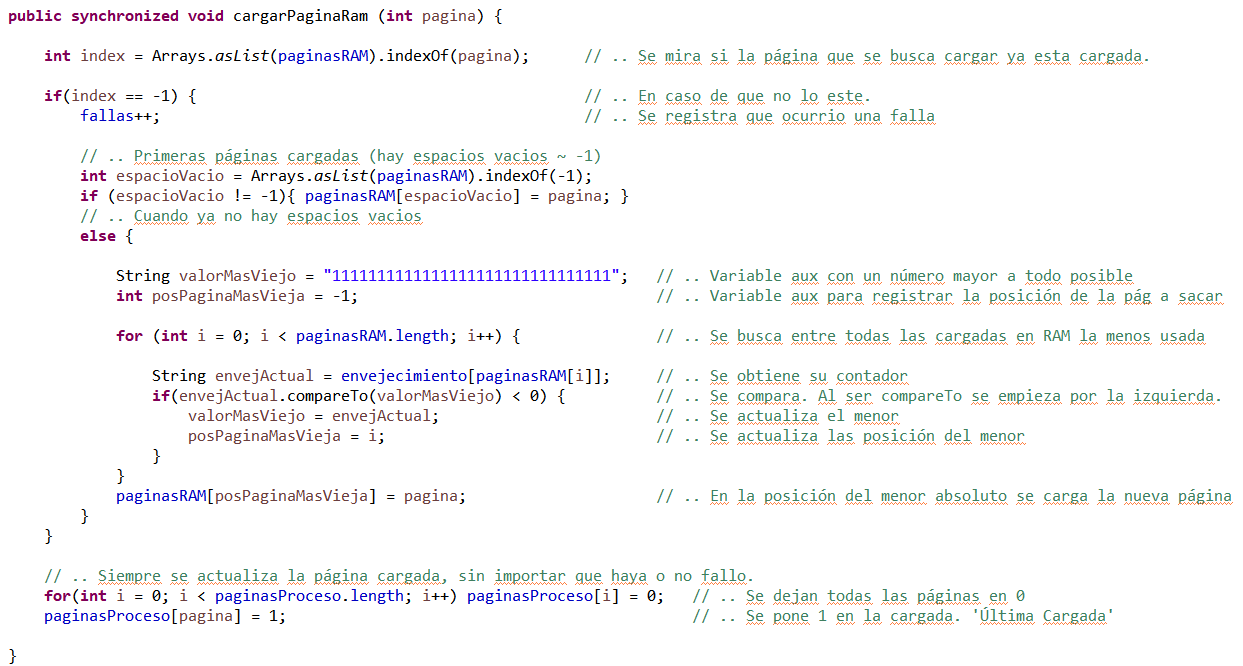
Como se observa se tiene igualmente un contador de las fallas.

Estas estructuras interactúan en dos métodos principales. El primero es actualizar el contador y el segundo cargar una página. Como ambos comparten estructuras están sincronizados sobre el mismo monitor.

En ese orden de ideas, el método de envejecer los contadores no recibe ningún parámetro y para cada página actualiza su valor dependiendo si fue cargada de últimas o no. Es decir, el arreglo de ‘paginasProceso’ contiene un único 1 en la posición de la página cargada de últimas y el resto son 0. Este valor siempre se pone a la izquierda, desplazando el resto para la derecha. Como se está haciendo un manejo de Strings, es solo concatenar por la izquierda y siempre mantener los primero 30 caracteres; en referencia a los 30 bits históricamente más recientes y significativos.



El método encargado de cargar una página consta de varios pasos, de los cuales uno siempre pasa. Apenas llega una página a cargar se revisa si la página ya está o no entre las cargadas en la RAM. Si ya está solo se procede a actualizar el arreglo ‘paginasProceso’ con el fin de que cuando el algoritmo de envejecer sea llamado registre correctamente la última usada. Por otro lado, si existe fallo de página, se aumenta la cuenta y se procede a realizar el intercambio con la página menos usada históricamente. Cabe resaltar que hay dos tipos de falla. Cuando hay espacios libres en la RAM y solo se va a cargar las páginas por primera vez, y cuando ya toca hacer intercambio. En el escenario uno solo se registra en la RAM la página a cargar y nuevamente se actualiza ‘paginasProceso’ (como siempre). Bajo el segundo escenario, se debe recorrer todas las páginas de la RAM y ver cuál es la menos usada históricamente. Este proceso se hace por medio de un recorrido y comparaciones de Strings por medio de ‘compareTo’. Esta comparación es válida y funcional para el caso debido a que ‘compareTo’ va carácter por carácter de izquierda a derecha. Es decir, compara por el bit más significativo, eligiendo así siempre la página menos usada. Luego se genera el intercambio y la debida actualización de la información.



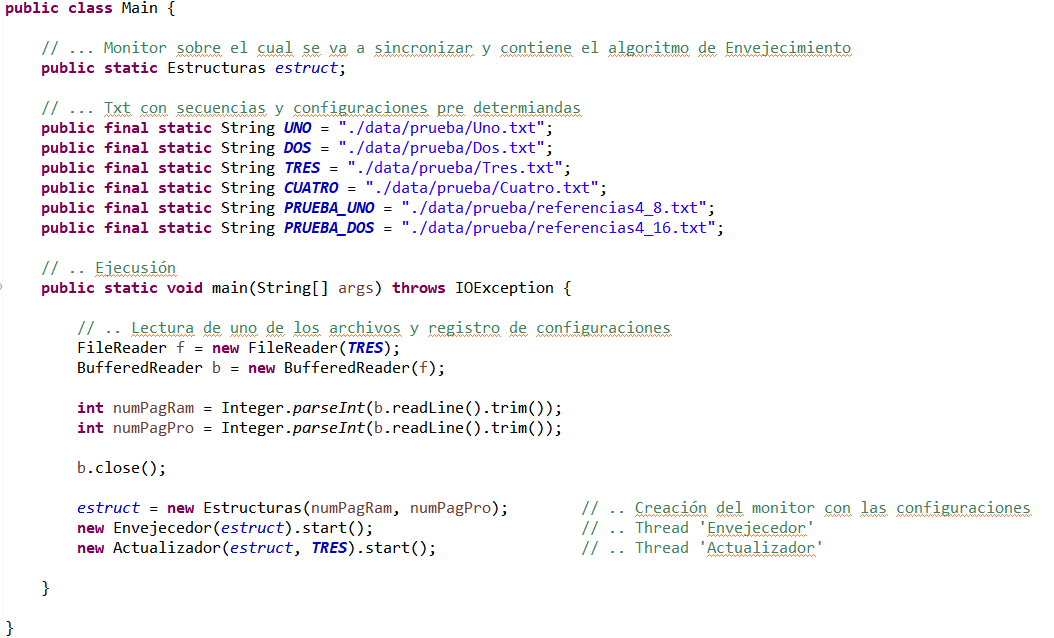
Con estos métodos y las estructuras ya presentadas, se logra ejecutar efectiva y correctamente el conteo de fallas de páginas por medio del algoritmo de envejecimiento.

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

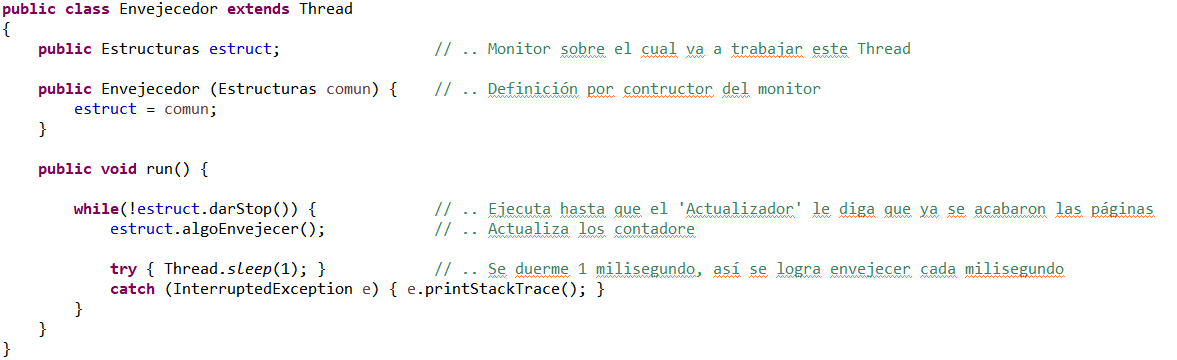
Para la sincronización de hace uso de dos Threads: ‘Envejecedor’ y ‘Actualizador’. Como su nombre lo indica, el primero ejecuta el método en envejecer y el segundo el de cargar/actualizar la página que llega a la RAM.

En ese orden de ideas, específicamente el ‘Envejecedor’ solo actualiza los contadores cada milisegundo; mientras que ‘Actualizador’, actualiza la referencia, registra el fallo de página si hay, al igual que la generación del intercambio una vez determino la menos usada históricamente. Este segundo ocurre cada 5 milisegundos como se especificó en las instrucciones.

Ambos Threads actúan bajo el mismo monitor y el ‘Envejecedor’ actúa siempre y cuando haya páginas por cargar. Por tanto, ‘Actualizador’ es el encargado de detenerlo. La forma de que ambos actúen bajo el mismo monitor fue posible al instanciarlo en una clase ‘Main’ y pasárselo como parámetro a ambas estructuras con las especificaciones previstas en un txt correspondiente. A continuación, se ve esta clase, la inicialización del monitor con sus configuraciones y el debido arranque de cada Thread. Se define que por archivo se necesita solo uno de cada tipo.

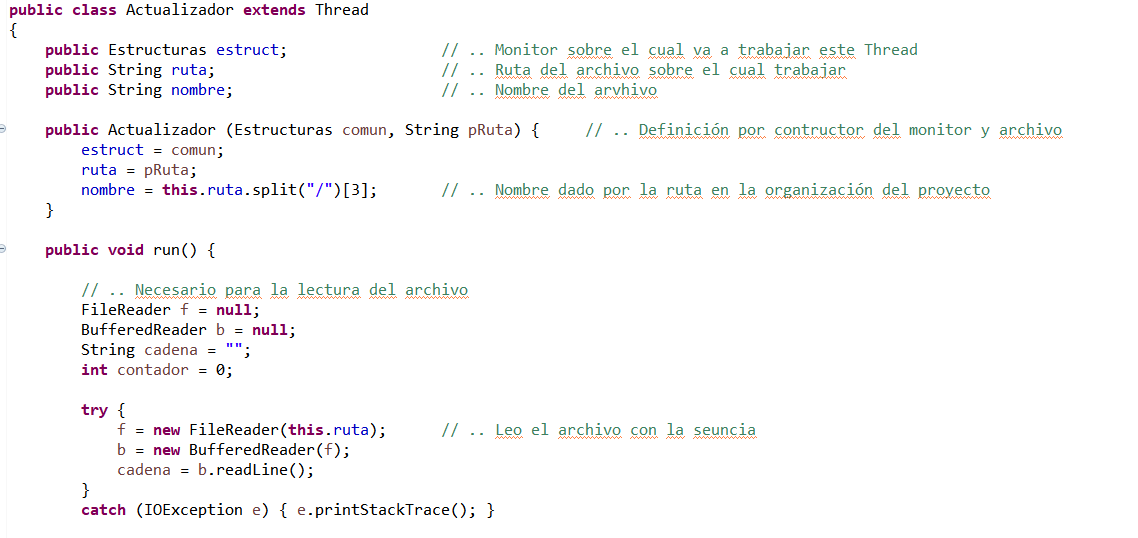


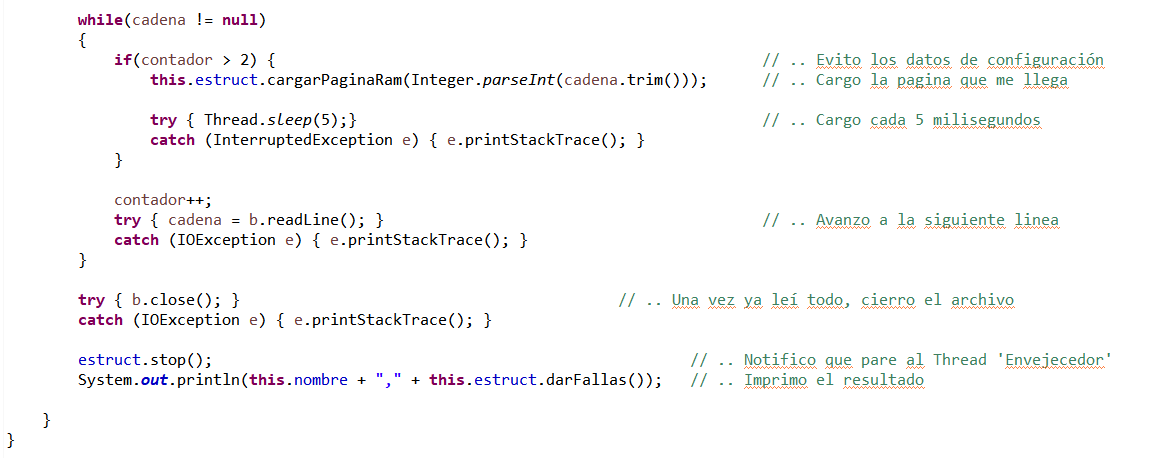
Hablando a detalle de cada Thread, se pueden detallar varios puntos. El Thread ‘Envejecedor’ es relativamente simple. Se crea y parametriza a su monitor por medio del constrictor. Para cuando el ‘Actualizador’ de la orden, y mientras, cada milisegundo actualiza los contadores del algoritmo de envejecimiento.



El Thread ‘Actualizador’ es más complejo en cuanto a implementación, pero igual de sencillo en ejecución. En un inicio cuenta con su monitor para trabajar, la ruta del archivo del cual va a leer la secuencia, y un nombre para imprimir de manera ordenada. El monitor y la nota se configurar por constructor y el nombre es una posición relativa con base a como estar guardados los txt’s del proyecto.

Una vez ya está configurado el monitor este empieza su ejecución de abriendo el archivo, omitiendo los datos de configuración y empezando a leer la secuencia. Cada dato de la secuencia es cargado en intervalos de 5 milisegundos. Esto se hace hasta que toda la secuencia haya sido cargada. Una vez acaba, se le avisa al Thread ‘Envejecer’ que puede parar y se imprime el número de fallas.

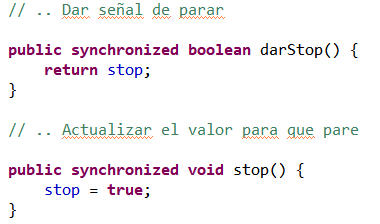




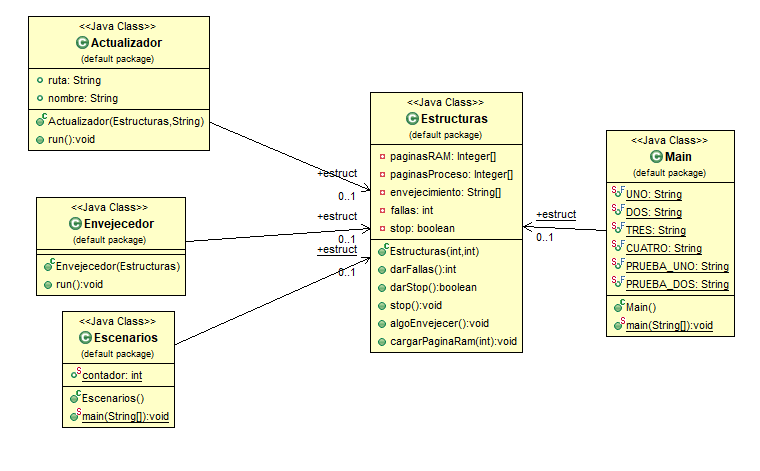
Para generar el stop se debe aclarar que se hace uso de la siguiente variable presente en el monitor.



El ‘Envejecedor’ de manera sincrónica constantemente pide su valor, y una vez fue cambiado a ‘true’ se detiene. Esta actualización se hace por medio de los siguientes métodos presentes en el monitor.



Por último, se presenta el modelo del UML del proyecto generado. La clase ‘escenarios’ se generó con el fin de realizar los datos a usar en las gráficas solicitadas y sigue la misma lógica ya explicada, pero para más de un archivo.



## Una tabla con los datos recopilados (número de fallas de página por cada caso simulado).

**NOTA:** Si bien en todos se logró obtener el valor esperado, este valor no fue constante tras cada iteración. Cada archivo presento resultados ligeramente por encima y por debajo. No obstante, ningún desfase es significativo como para concluir que el algoritmo está mal. Estos desfases se deben a la sincronización. Una explicación a esto es que como el envejecedor corre cada milisegundo y el actualizador cada cinco milisegundos, es posible que cada cinco ejecuciones del envejecedor el actualizador quiera acceder al monitor, por lo que en ocasiones podrían no haber exactamente cinco ejecuciones del algoritmo de envejecimiento, sino 4 o incluso 6 antes de que el actualizador pueda acceder a los recursos de estructuras que están protegidos.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Archivo** | **Esperado** | **Mínimo** | **Máximo** | **Desfase inferior** | **Desfase superior** | **Mejor** |
| referencias1.txt | 587 | 586 | 594 | -1 | 7 | 587 |
| referencias2.txt | 577 | 573 | 577 | -4 | 0 | 577 |
| referencias3.txt | 758 | 755 | 758 | -3 | 0 | 758 |
| referencias4.txt | 340 | 325 | 347 | -15 | 7 | 340 |
| referencias4\_8.txt | 5 | 5 | 5 | 0 | 0 | 5 |
| referencias4\_16.txt | 12 | 12 | 12 | 0 | 0 | 12 |

## Una serie de gráficas que ilustren el comportamiento del sistema. Para eso cree una gráfica por cada tamaño de programa estudiado y en cada una ilustre: número de fallas de página vs. número de marcos asignados vs. nivel de localidad.

Las gráficas que se presentan a continuación se generaron gracias a los archivos ‘Generador.py’ y ‘Graficas.py’. El primero es un script que genera archivos txt con base a parámetros definidos. El proceso/algoritmo que lo genera está debidamente comentado en mismo archivo, y se usó para variar los tres parámetros solicitados en diferentes combinaciones. Un ejemplo de los archivos generados está en la carpeta ‘escenarios’ en ‘data’.

Una vez se tenían estos archivos se procedió a correr la clase ‘Escenarios’ del proyecto. Cabe resaltar que se corrió solo un tipo de archivo a la vez y se copió manualmente su resultado el txt predispuesto para esa labor. Este txt fue leído por el segundo archivo de Python el cual se encarga de agrupar, ordenar y graficar los resultados correspondientes. A continuación, se presentan las gráficas obtenidas posterior a la variación de los parámetros:

**Número de Marcos Asignados:Gráfico, Gráfico de líneas, Gráfico de dispersión

Descripción generada automáticamenteGráfico, Gráfico de líneas, Gráfico de dispersión

Descripción generada automáticamenteGráfico, Gráfico de líneas, Gráfico de dispersión

Descripción generada automáticamente**

### **Número de Páginas del Proceso**

**Gráfico, Gráfico de dispersión

Descripción generada automáticamenteGráfico, Gráfico de líneas, Gráfico de dispersión

Descripción generada automáticamenteGráfico, Gráfico de líneas, Gráfico de dispersión

Descripción generada automáticamente**

### **Nivel de LocalidadGráfico, Gráfico de líneas Descripción generada automáticamenteGráfico, Gráfico de líneas Descripción generada automáticamenteGráfico, Gráfico de líneas Descripción generada automáticamente**

## Escriba su interpretación de los resultados: ¿tienen sentido? Justifique su respuesta.

Como se pudo observar, con el total de 9 gráficas, cada parámetro tiene repercusiones significativas en el desempeño del algoritmo. En primera instancia se destaca que para los parámetros asociados a páginas se hizo uso de tres niveles de localidad diferente; y para el nivel de localidad, se hizo uso de tres diferentes números de marcos asignados. A continuación, la interpretación de estos resultados.

### Número de Marcos Asignados

El número de fallos en relación con este factor se puede observar que claramente tiene un comportamiento decreciente, de relación inversa. Es decir, a mayor cantidad de número de marcos asignados menor es la cantidad de fallas. Esto tiene sentido debido a que más paginas estar cargadas y por tanto serán pocas aquellas que generen un fallo. La segunda relación es con respecto a nivel de localidad. Se puede notar que cuando estos valores son bajos, la cantidad de fallos cae de manera cóncava. Mientras que cuando el nivel tiene a 1, el comportamiento se presenta casi de manera lineal. Esto se explica, debido a que cuando el nivel de localidad es bajo la secuencia de repetidos se concentra en unos pocos; una vez esos pocos se cargan, el número de fallos se reduce drásticamente. Pero, cuando la secuencia de repetidos se distribuye en muchos valores de las páginas del proceso es normal que sigan ocurriendo fallas, ya que se va a demandar cambiar las paginas constantemente. La tendencia de estas graficas muestran que cuando haya más marcos asignados que páginas del proceso el número de fallas será mínimo.

### Número de Páginas del Proceso

El número de fallas frente a la cantidad de páginas de proceso sigue un comportamiento interesante. A primera vista están claramente directamente relacionados, es decir, si uno sube el otro también. No obstante, esta subida no es de comportamiento lineal, sino un poco logarítmica. En otras palabras, empieza creciente de manera pronunciada pero luego de cierto valor se estabiliza. Esto es razonable debido a que, con número extremo de páginas del proceso, el algoritmo tiende a estabilizarse y sacar de manera controlada las menos usadas. Al insertar además la variable de nivel de localidad, esto cobra sentido al ver que, entre más distribuidos es la repetición de los datos en la secuencia más rápido de alcanza la estabilidad. Mientras que, entre menos repetidos, más se demora. Visto de otra manera, si los repetidos son pocos y los no repetidos muchos, lograr cubrir todos los no repetidos llevara a que las fallas parezcan crecer lineal hasta que se acaban y luego si se estabiliza. Pero el tiempo en el que se acaban causa que no converjan rápido. Por el contrario, si los repetidos son muchos, los no repetidos causaran un crecimiento abrupto en las fallas mientras se cubren y luego convergerá de manera rápida.

### Nivel de Localidad

El número de fallos en relación con el nivel de localidad presenta un comportamiento parecido a lineal, donde están positivamente relacionados. Si uno sube, el otro también. El punto de inflexión muestra aquel porcentaje donde una vez superado el número de fallas aumenta de manera drástica. Al insertar la variabilidad en el número de marcos asignados, el comportamiento pasa a ser entre cóncavo a lineal. Entre menos es la cantidad de marcos más efecto tiene el nivel de localidad una vez pasa el punto de inflexión. Entre más tiene, menos pronunciado se hace el crecimiento post punto crítico. Esto tiene sentido al entender que entre más dispersos los datos (la secuencia de repetidos se encuentra distribuida entre más páginas del proceso), más intercambios deberá hacerse con la RAM al momento de cargar las paginas, porque es menos probable que la página que se está cargando este. Igualmente, es lógico ver que al tener mayores marcos asignados el crecimiento abrupto se suavice. Esto se debe a que se aumenta la posibilidad de que la pagina este cargada y, por tanto, el comportamiento más lineal.