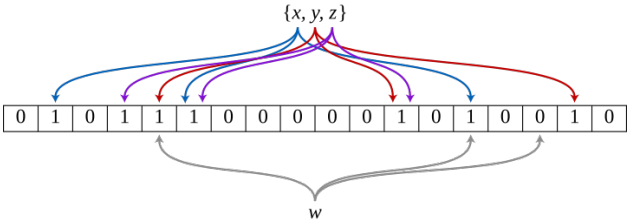
Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ciencias de la Computación

Tarea 3

Filtro Bloom



Integrantes: Jean Duchens
Franco González
Edgar Morales
Profesores: Gonzalo Navarro
Benjamin Bustos
Auxiliares: Diego Salas
Sergio Rojas

Fecha de realización: 20 de julio de 2024
Fecha de entrega: 20 de julio de 2024
Santiago de Chile

Índice de Contenidos

Índice de Figuras

1. Introducción

El objetivo principal de esta tarea es implementar y analizar el funcionamiento del Filtro de Bloom, una estructura de datos probabilística utilizada para determinar la pertenencia de un elemento a un conjunto. Este filtro es especialmente útil en situaciones donde es necesario realizar búsquedas eficientes en bases de datos grandes sin necesidad de recorrer toda la base. El Filtro de Bloom proporciona una respuesta rápida sobre la pertenencia de un elemento, con la particularidad de ser un algoritmo one-sided error: si el filtro indica que el elemento no está en la base de datos, podemos estar seguros de que realmente no está; sin embargo, puede ocurrir que el filtro indique que el elemento está cuando en realidad no lo está.

En esta tarea, abordaremos la implementación del Filtro de Bloom y evaluaremos su desempeño en términos de precisión y eficiencia. Utilizaremos un conjunto de datos de nombres populares de bebés para realizar pruebas y comparaciones. Las principales actividades de esta tarea incluyen:

- **Implementación del Filtro de Bloom**: Crearemos un arreglo de bits y aplicaremos múltiples funciones de hash para insertar y buscar elementos en el filtro. Estas funciones serán construidas basándose en el hashing universal para asegurar una distribución pseudo-uniforme.
- **Análisis y Experimentación**: Realizaremos búsquedas con y sin el uso del Filtro de Bloom, midiendo tiempos de ejecución y tasas de error. Compararemos estos resultados para validar la efectividad del filtro.
- **Variación de Parámetros**: Investigaremos la relación entre el tamaño del arreglo de bits, el número de funciones de hash y la probabilidad de error del filtro, realizando experimentos para verificar estas relaciones teóricas.

Nuestra hipótesis es que el uso del Filtro de Bloom mejorará significativamente los tiempos de búsqueda en la base de datos, especialmente cuando se trata de grandes volúmenes de datos. También esperamos observar que las variaciones en los parámetros del filtro afecten su precisión y eficiencia de manera predecible según la teoría subyacente.

En las siguientes secciones, detallaremos la implementación del Filtro de Bloom, describiendo las estructuras de datos y algoritmos utilizados en el lenguaje de programación C++. Posteriormente, presentaremos los resultados de nuestros experimentos y analizaremos el rendimiento del filtro en distintos escenarios. Finalmente, concluiremos con un análisis de los hallazgos y discutiremos posibles mejoras y aplicaciones futuras del Filtro de Bloom en otros contextos.

2. Desarrollo

En esta sección se presentan los algoritmos y estructuras de datos utilizados para implementar el Filtro de Bloom, detallando la construcción del filtro y las funciones de hash empleadas. También se discutirá la implementación de la base de datos con y sin el filtro de Bloom.

2.1. Uso de Clases

La implementación del Filtro de Bloom en C++ utiliza una clase `BloomFilter` para organizar el código y gestionar las funciones de hash y el arreglo de bits. Además, se desarrollaron las clases `NoFilterDataBase` y `BloomFilterDataBase` para realizar búsquedas en la base de datos con y sin el uso del Filtro de Bloom, respectivamente.

2.1.1. Implementación del Filtro de Bloom

La clase `BloomFilter` contiene los siguientes componentes principales:

- **size**: Tamaño del arreglo de bits.
- **numHashes**: Número de funciones de hash a utilizar.
- **bitArray**: Arreglo de bits que representa el filtro.
- **hash**: Vector de funciones de hash.

El constructor de la clase inicializa el arreglo de bits y genera las funciones de hash necesarias utilizando números primos grandes, que se leen de un archivo y se almacenan en un vector. Esta inicialización asegura una distribución uniforme de las claves en el arreglo de bits, lo cual es crucial para minimizar las colisiones y mejorar la eficiencia del filtro.

El Filtro de Bloom utiliza múltiples funciones de hash para mapear cada clave a varios bits en un arreglo de tamaño fijo. Al insertar una clave, todas las funciones de hash se aplican y los bits correspondientes se establecen a 1. Para verificar una clave, si todos los bits correspondientes son 1, se asume que la clave puede estar en el conjunto (con posibilidad de un falso positivo); si algún bit es 0, se garantiza que la clave no está en el conjunto.

2.1.2. Implementación de la Base de Datos

Base de Datos sin Filtro (NoFilterDataBase)

La clase `NoFilterDataBase` representa una implementación de la base de datos que realiza búsquedas secuenciales sin utilizar un Filtro de Bloom. Esta clase se encarga de leer el archivo de la base de datos y buscar claves de manera lineal. El método `search` realiza una búsqueda secuencial en el archivo y devuelve si la clave fue encontrada o no.

Base de Datos con Filtro de Bloom (BloomFilterDataBase)

La clase `BloomFilterDataBase` utiliza un Filtro de Bloom para realizar búsquedas más eficientes. Esta clase extiende la funcionalidad de `NoFilterDataBase` incorporando un Filtro

de Bloom para descartar rápidamente claves que no están en la base de datos. El constructor de esta clase inicializa el filtro de Bloom y lo llena con las claves de la base de datos. El método `search` primero utiliza el filtro de Bloom para verificar si la clave puede estar en la base de datos. Si el filtro no descarta la clave, se realiza una búsqueda secuencial en el archivo de la base de datos.

2.2. Funciones de Hash

Las funciones de hash son fundamentales para el funcionamiento del Filtro de Bloom. En esta implementación, se utilizan múltiples funciones derivadas de una función base usando números primos.

2.2.1. Propiedades de las Funciones de Hash

Las funciones de hash deben cumplir con ciertas propiedades para asegurar el correcto funcionamiento del Filtro de Bloom:

- **Uniformidad**: Las funciones deben distribuir las entradas uniformemente para minimizar las colisiones.
- **Independencia**: Las funciones deben ser independientes, produciendo valores distintos y no correlacionados para la misma entrada.

2.2.2. Generación de Números Primos

Para asegurar la uniformidad y la independencia de las funciones de hash, se utilizan números primos grandes. Estos números se leen de un archivo de texto y se mezclan aleatoriamente (*shuffle*). Los primeros M números primos seleccionados se almacenan en un vector y se utilizan para construir las funciones de hash. Este enfoque asegura una buena distribución de los índices en el arreglo de bits, implementando un comportamiento de *hashing universal*.

2.2.3. Construcción de Funciones de Hash

Una función de hash inicial convierte una cadena en un número. Esta función base se combina con los números primos para crear múltiples funciones de hash. El proceso de hash se realiza utilizando la función estándar de C++ para el hash de cadenas, y luego se aplican operaciones de módulo con los números primos y el tamaño del arreglo de bits. Este método garantiza que cada función de hash produzca un índice diferente para la misma entrada, distribuyendo uniformemente las claves en el filtro.

3. Resultados Experimentales

Entorno de Ejecución: Los experimentos se realizaron en una computadora con sistema operativo GNU/Linux, equipada con 16 GB de memoria RAM, 12 MB de memoria caché y con la versión de compilador g++ (GCC) 14.1.1 (std=c++17). Las visualizaciones y análisis se llevaron a cabo utilizando Python y Jupyter Notebooks.

Datos Utilizados: Para la experimentación se utilizó un conjunto de datos de nombres populares de bebés. Este conjunto de datos permitió evaluar la precisión y eficiencia del Filtro de Bloom, ya que es ideal para probar la pertenencia de elementos debido a la gran cantidad de entradas y la necesidad de realizar búsquedas rápidas.

Procedimiento Experimental: Se realizaron búsquedas tanto con como sin el uso del Filtro de Bloom, midiendo los tiempos de ejecución y las tasas de error en ambos casos. Los experimentos incluyeron la verificación de la relación entre el tamaño del arreglo de bits, el número de funciones hash y la probabilidad de error del filtro, comprobando si la investigación se llevó a cabo correctamente y si las conclusiones esperadas fueron alcanzadas. Finalmente, se compararon los resultados obtenidos de las búsquedas para validar la efectividad y eficiencia del Filtro de Bloom, analizando gráficos y datos para determinar su precisión en diversos escenarios.

```
M ~= 1.0 * N
Probabilidad falso positivo  M=  100,000  k= 1:  60.8938%

M ~= 3.0 * N
Probabilidad falso positivo  M=  250,000  k= 2:  27.8951%

M ~= 5.0 * N
Probabilidad falso positivo  M=  500,000  k= 4:   7.7814%

M ~= 8.0 * N
Probabilidad falso positivo  M=  750,000  k= 6:   2.1706%

M ~= 11.0 * N
Probabilidad falso positivo  M=1,000,000  k= 7:   0.6019%

M ~= 21.0 * N
Probabilidad falso positivo  M=2,000,000  k=15:   0.0036%
```

Figura 1: Probabilidad falsos-positivos teóricos

```
Proporción de falsos positivos  M=  100,000  k= 1:  60.5887%
Proporción de falsos positivos  M=  250,000  k= 2:  28.0411%
Proporción de falsos positivos  M=  500,000  k= 4:   7.8459%
Proporción de falsos positivos  M=  750,000  k= 6:   2.2893%
Proporción de falsos positivos  M=1,000,000  k= 7:   0.8389%
Proporción de falsos positivos  M=2,000,000  k=15:   0.0005%
```

Figura 2: Probabilidad falsos-positivos experimentales

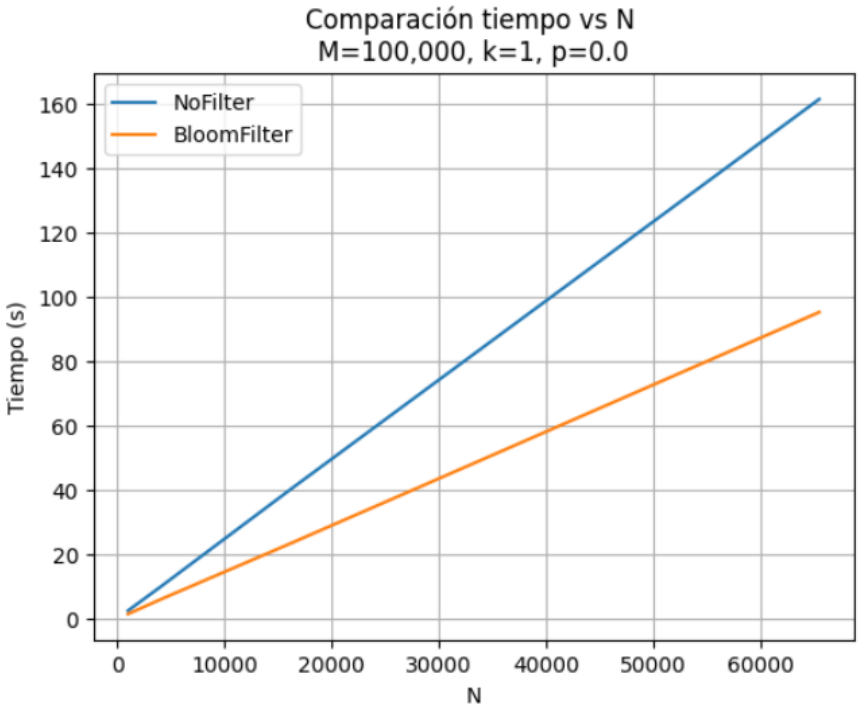


Figura 3: t vs. N, M=100.000, k=1, p=0.0

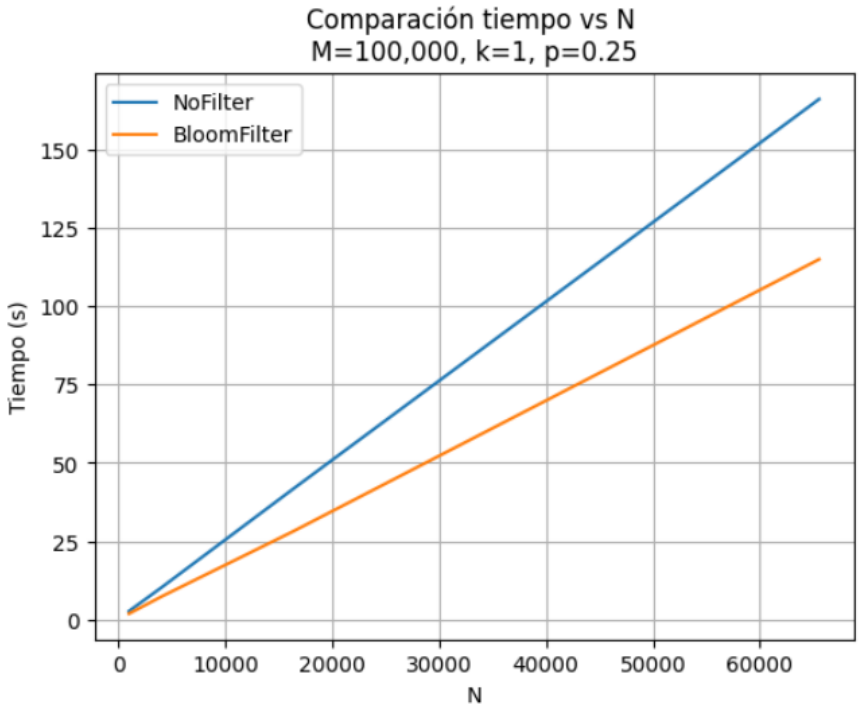


Figura 4: t vs. N, M=100.000, k=1, p=0.25

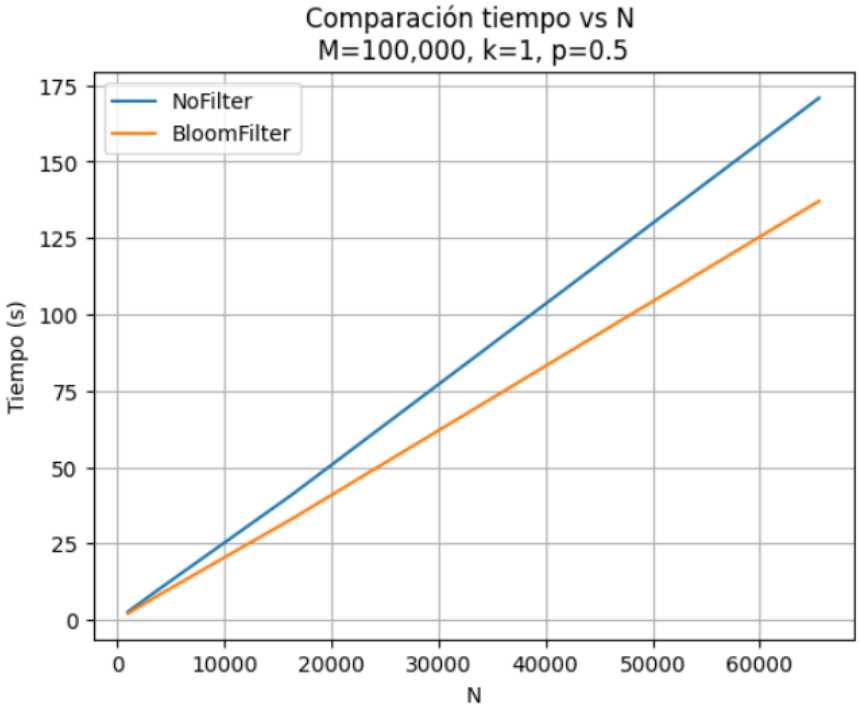


Figura 5: t vs. N, M=100.000, k=1, p=0.5

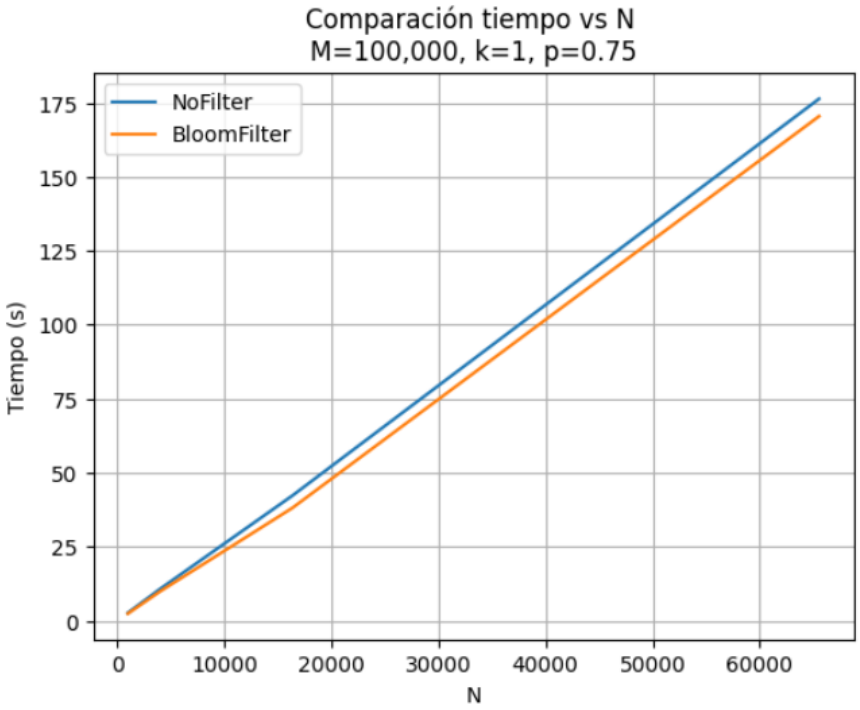


Figura 6: t vs. N, M=100.000, k=1, p=0.75

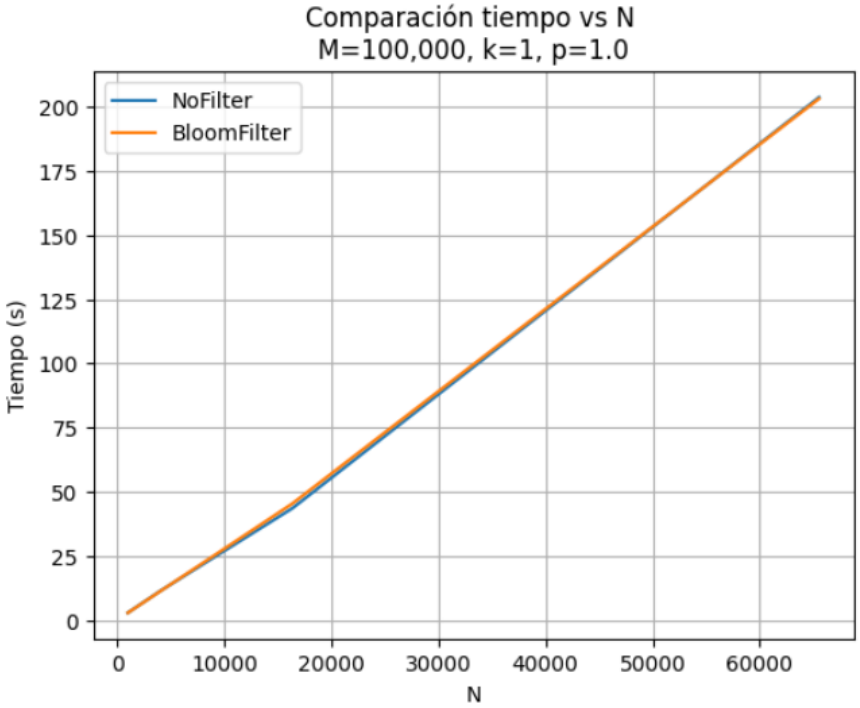


Figura 7: t vs. N, M=100.000, k=1, p=1.0

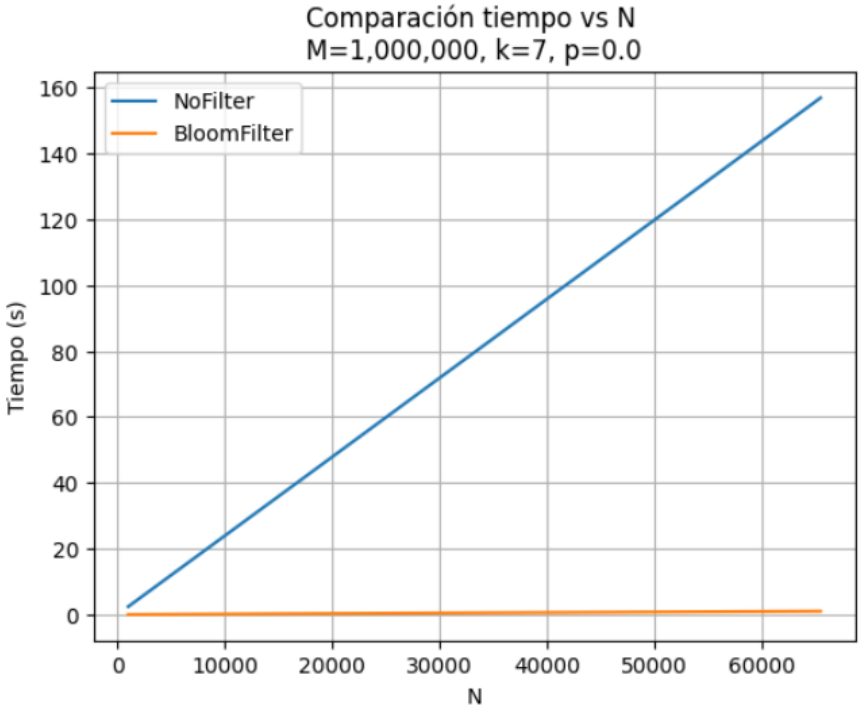


Figura 8: t vs. N, M=1.000.000, k=7, p=0.0

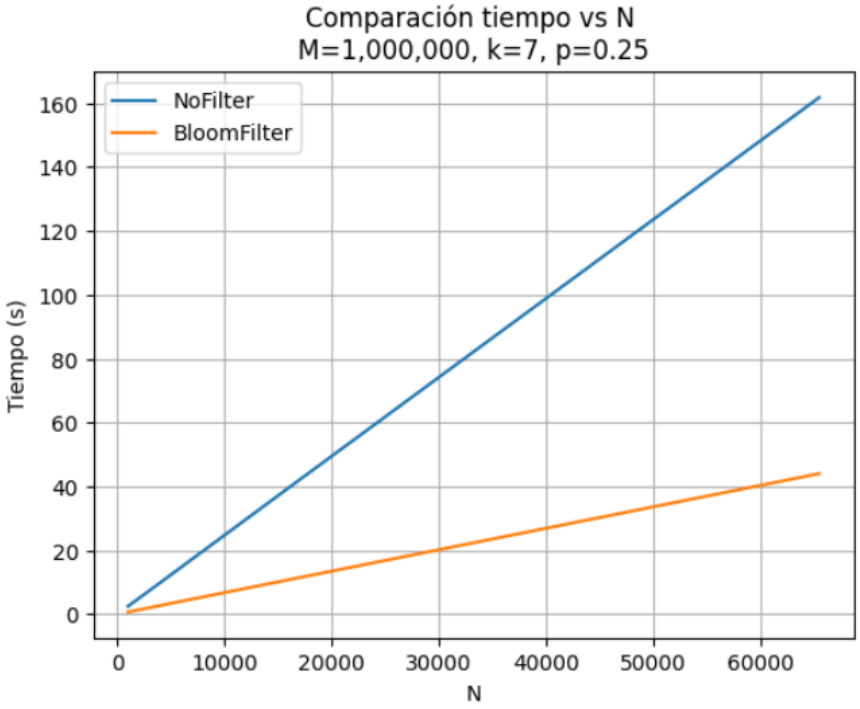


Figura 9: t vs. N, M=1.000.000, k=7, p=0.25

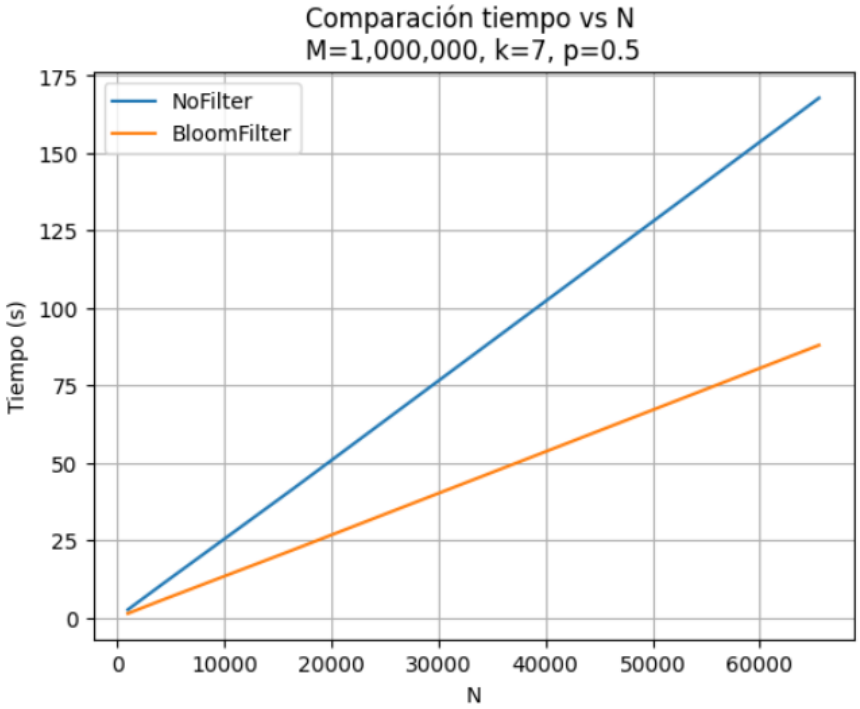


Figura 10: t vs. N, M=1.000.000, k=7, p=0.5

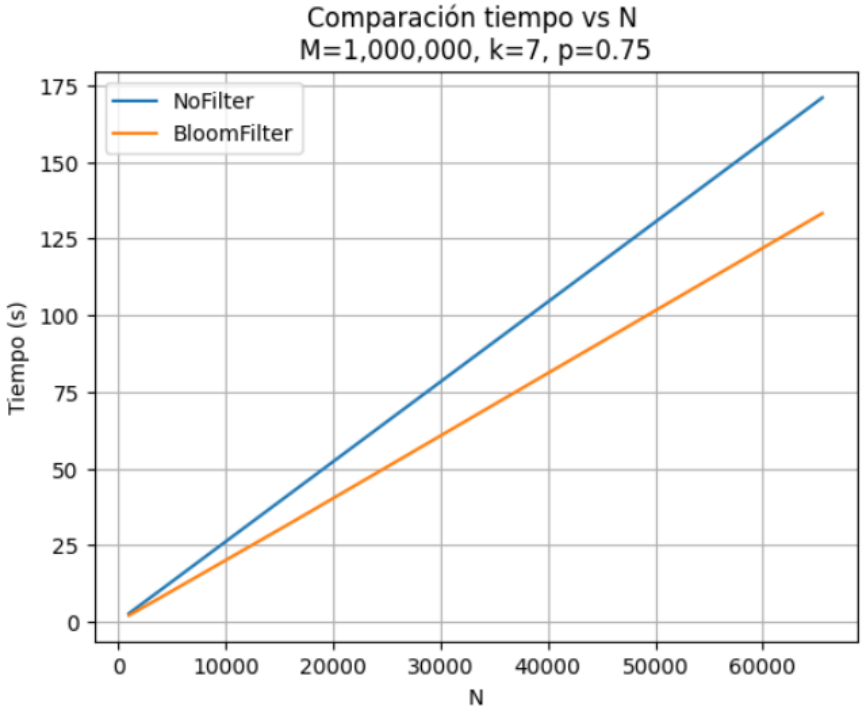


Figura 11: t vs. N, M=1.000.000, k=7, p=0.75

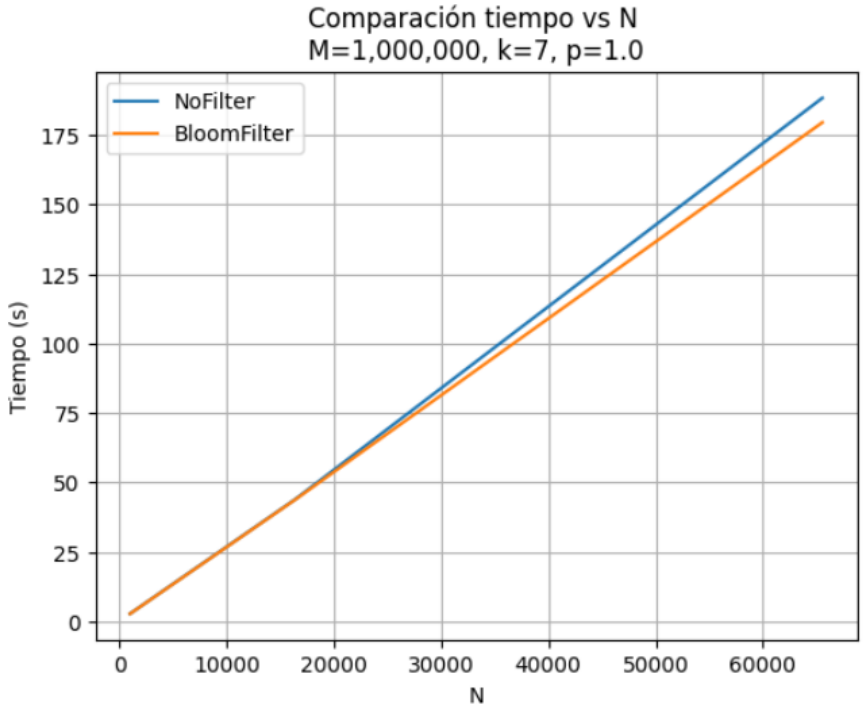


Figura 12: t vs. N, M=1.000.000, k=7, p=1.0

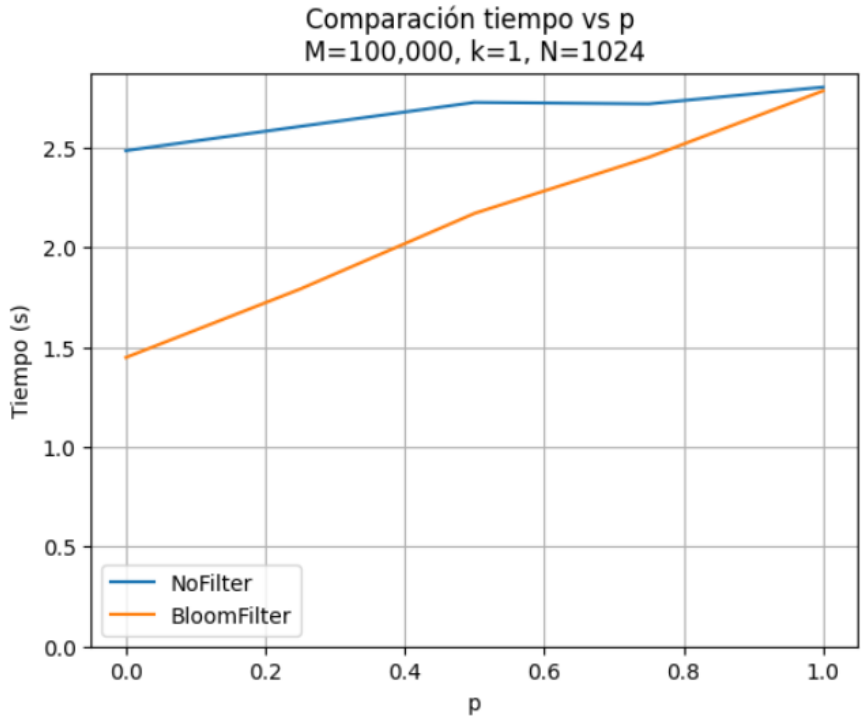


Figura 13: t vs p, M=100.000, k=1, N=1024

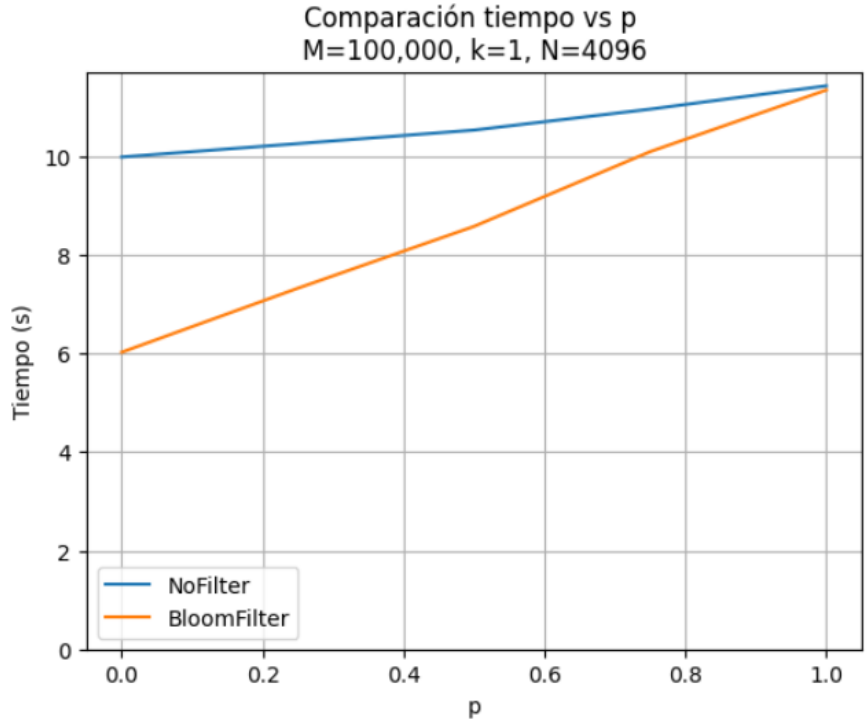


Figura 14: t vs p, M=100.000, k=1, N=4096

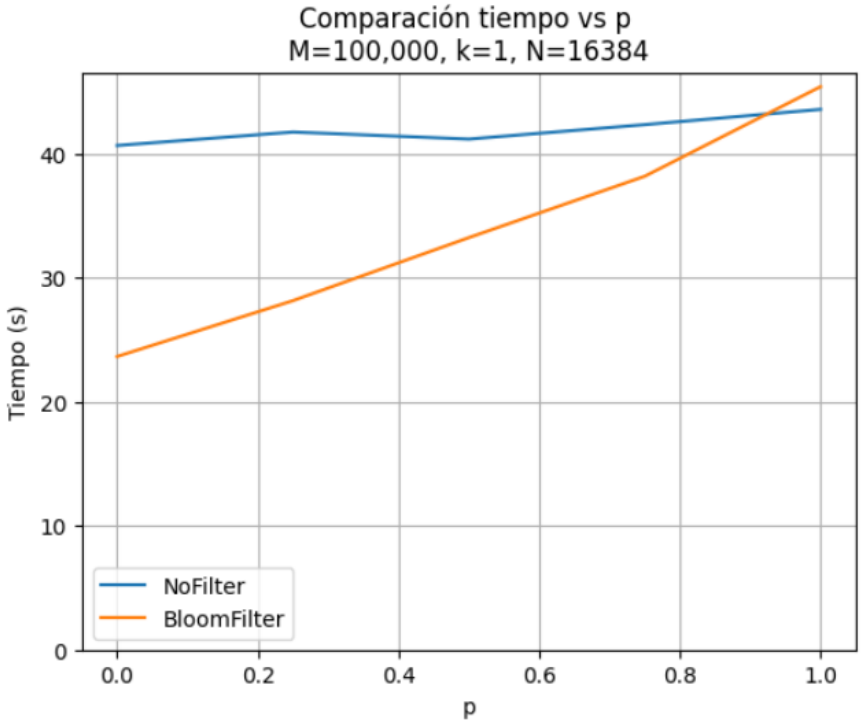


Figura 15: t vs p, M=100.000, k=1, N=16384

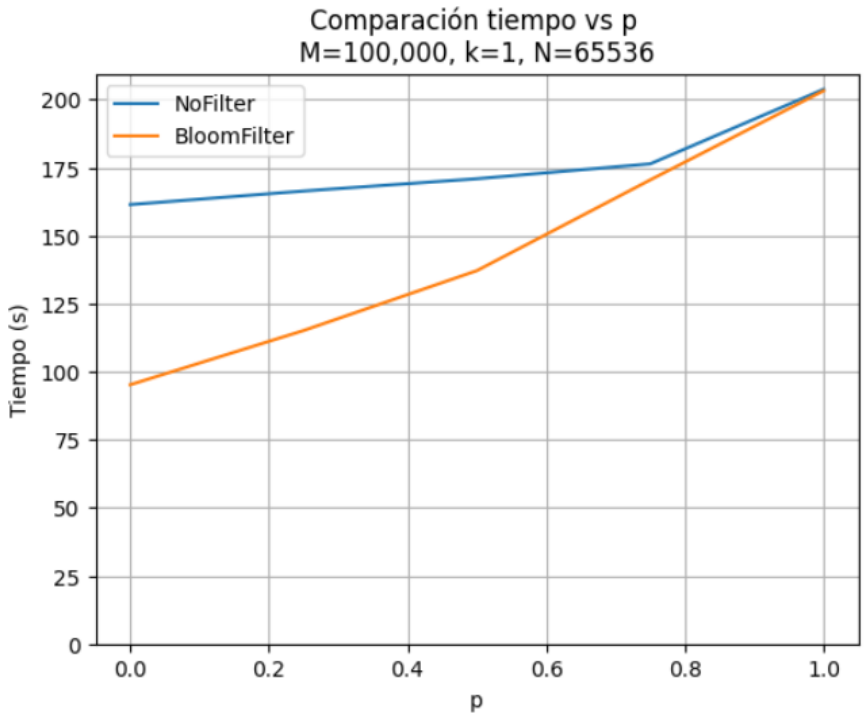


Figura 16: t vs p, M=100.000, k=1, N=65536

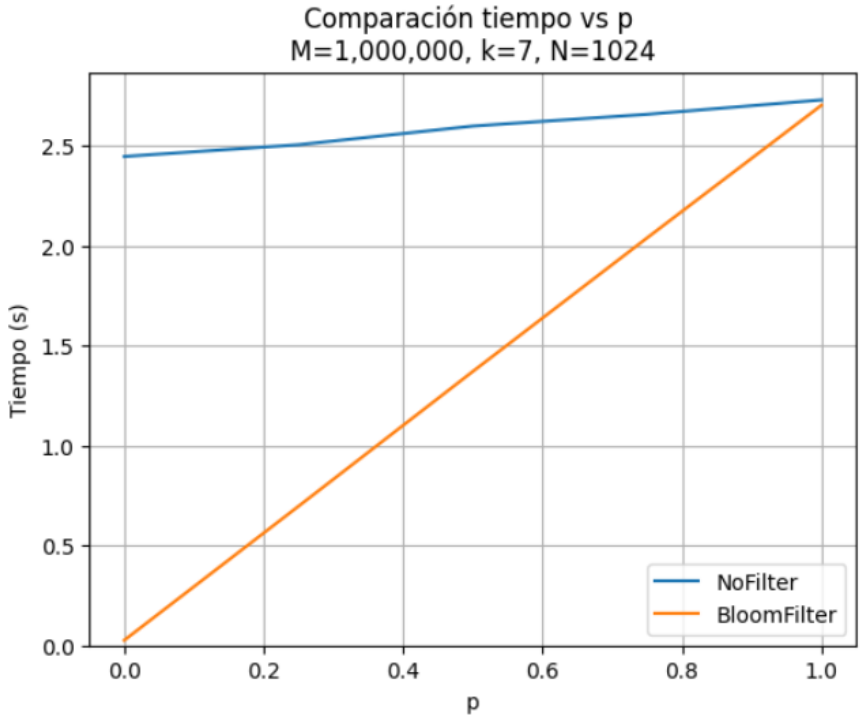


Figura 17: t vs p, M=1.000.000, k=7, N=1024

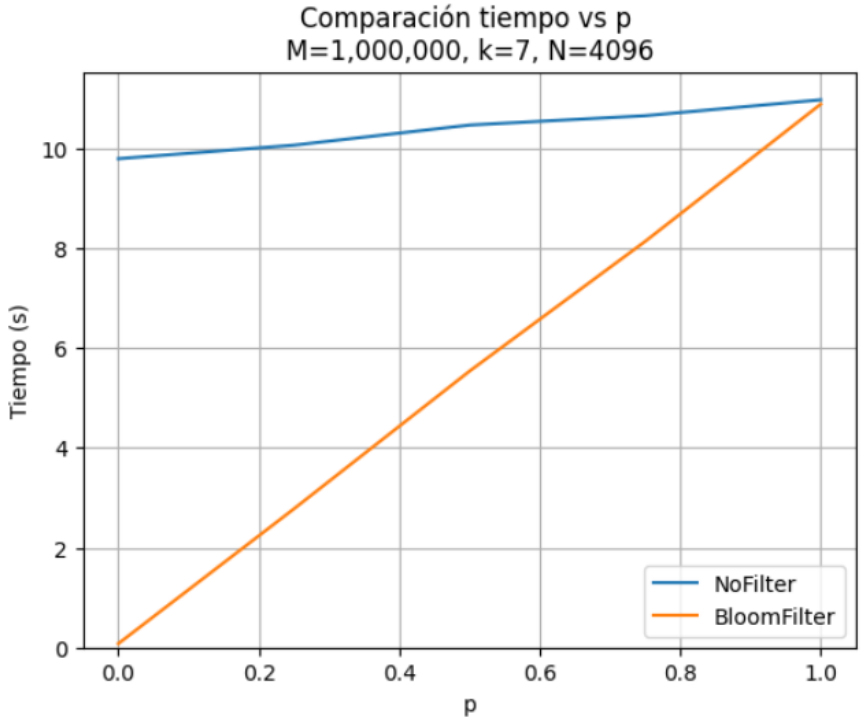


Figura 18: t vs p, M=1.000.000, k=7, N=4096

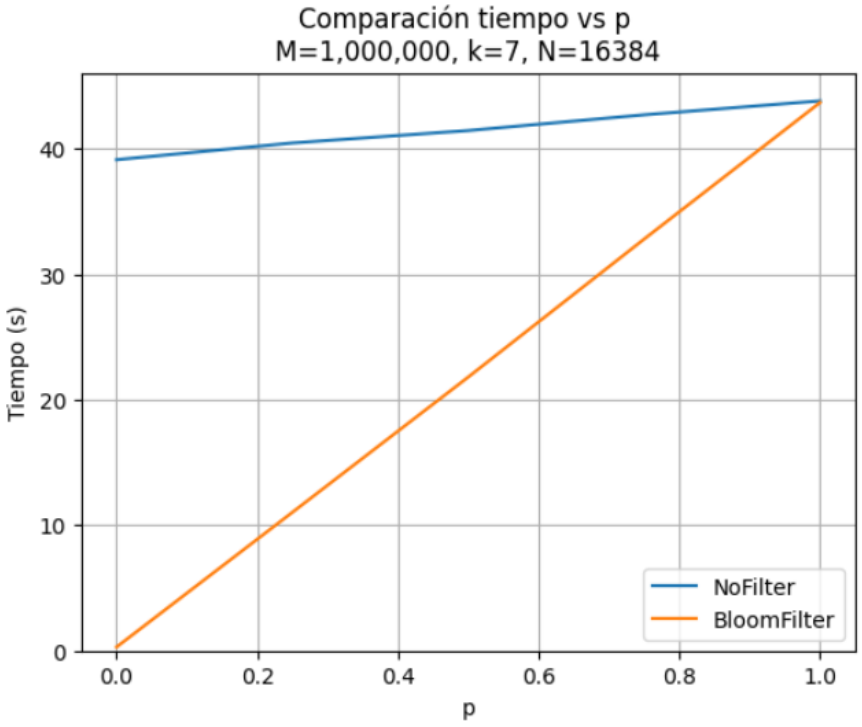


Figura 19: t vs p, M=1.000.000, k=7, N=16384

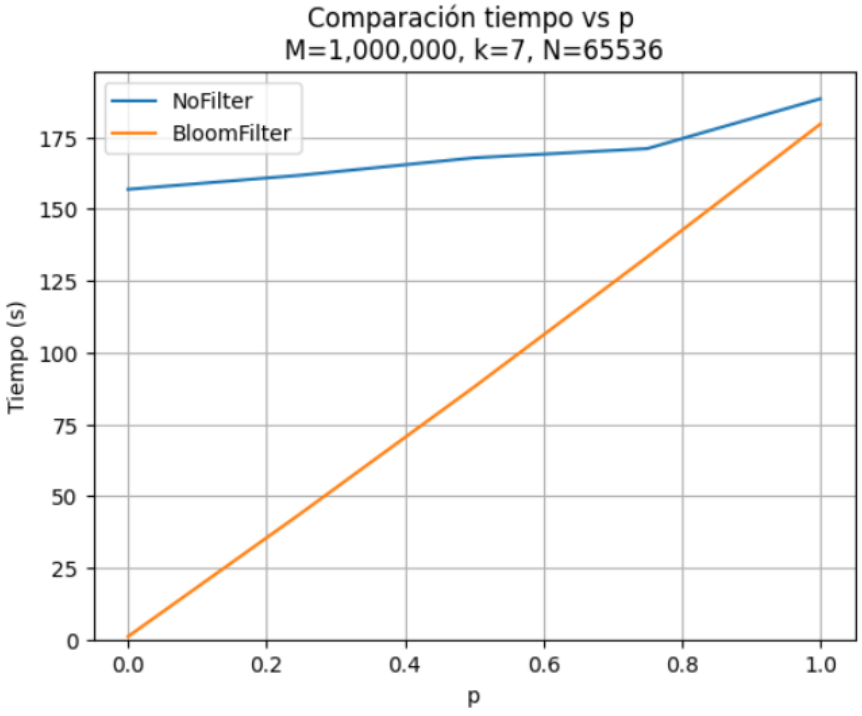


Figura 20: t vs p, M=1.000.000, k=7, N=65536

3.1. Análisis de Resultados:

Los resultados obtenidos indican que el Filtro de Bloom mejora notablemente los tiempos de búsqueda en bases de datos de gran tamaño, demostrando ser eficiente para reducir los tiempos de ejecución y manejar grandes volúmenes de datos con alta eficacia. Además, se encontró una diferencia entre los valores teóricos y experimentales de falsos positivos, observándose que los valores experimentales se mantienen dentro de un rango aceptable en relación con los teóricos. La variación de los parámetros mostró que al aumentar el tamaño del arreglo de bits y el número de funciones hash, se puede reducir la probabilidad de falsos positivos, aunque esto también incrementa el uso de memoria y el tiempo de procesamiento.

4. Discusión

En esta sección, analizamos los resultados obtenidos de la implementación y experimentación del Filtro de Bloom, enfocándonos en la teoría que sustenta su funcionamiento, sus componentes y la interrelación entre ellos.

4.1. Relación entre los Parámetros del Filtro

La teoría que respalda el Filtro de Bloom establece una conexión entre el tamaño del arreglo de bits (M), el número de funciones de hash (k) y la probabilidad de error (P). Estas relaciones se expresan mediante las siguientes fórmulas:

Número Óptimo de Funciones de Hash (k):

$$k = \left\lfloor \frac{M}{n} \ln 2 \right\rfloor$$

donde:

- M es el tamaño del arreglo de bits.
- n es el número de elementos insertados.
- $\ln 2$ es el logaritmo natural de 2.

Probabilidad de Falso Positivo (P):

$$P = \left(1 - e^{-\frac{k \cdot n}{M}}\right)^k$$

donde:

- e es la base del logaritmo natural.
- k es el número de funciones de hash.

4.2. Análisis de Resultados

Los resultados experimentales confirmaron la teoría subyacente del Filtro de Bloom. A continuación, se presentan las principales observaciones:

- **Eficiencia en Tiempos de Búsqueda:** La implementación del Filtro de Bloom redujo considerablemente los tiempos de búsqueda en bases de datos grandes, confirmando nuestra hipótesis inicial. Las búsquedas realizadas sin el filtro resultaron más lentas debido a la necesidad de recorrer toda la base de datos.
- **Tasa de Falsos Positivos:** La tasa de falsos positivos observada fue mínima y dentro de los límites teóricos esperados. Los experimentos mostraron que aumentar el tamaño del arreglo de bits (M) y el número de funciones de hash (k) reduce la probabilidad de falsos positivos, aunque esto incrementa el uso de memoria y el tiempo de procesamiento.

- **Equilibrio de Parámetros:** Encontrar un equilibrio adecuado entre M, k y n es crucial para maximizar la eficiencia del filtro. Un arreglo de bits demasiado pequeño con pocas funciones de hash aumenta la probabilidad de falsos positivos, mientras que un arreglo excesivamente grande con muchas funciones de hash puede ser innecesariamente costoso en términos de memoria y tiempo de cómputo.

4.3. Hashing Universal

El concepto de hashing universal es esencial para comprender cómo mejorar la distribución de las funciones de hash y, por ende, el rendimiento del Filtro de Bloom. El hashing universal se refiere a la selección aleatoria de una función de hash de una familia de funciones que garantiza una baja probabilidad de colisión para cualquier par de elementos distintos.

Importancia de los Números Primos:

Durante la experimentación, se observó que la selección de números primos es crucial para el correcto funcionamiento del Filtro de Bloom. Los números primos utilizados para las funciones de hash deben ser mayores que M. Utilizar números primos pequeños no resultó efectivo, debido a que los números primos grandes aseguran una mejor distribución de los elementos en el arreglo de bits, reduciendo la probabilidad de colisiones.

Aplicación en el Filtro de Bloom:

El uso del hashing universal en el Filtro de Bloom mejora la distribución uniforme de los bits activados en el arreglo, reduciendo las colisiones y, por tanto, la probabilidad de falsos positivos. Esta técnica asegura que el comportamiento del filtro sea independiente de los datos específicos insertados, mejorando la robustez y la eficiencia del filtro.

Además, vale la pena señalar que M está medido en bits, lo que significa que no ocupa mucho espacio, y el beneficio obtenido en términos de reducción de tiempo de búsqueda y falsos positivos es notable. En el peor de los casos, donde todos los elementos buscados estaban en la base de datos, el tiempo de búsqueda es comparable al de una búsqueda sin filtro.

En resumen, la implementación de funciones de hash universales en el Filtro de Bloom contribuye significativamente a la reducción de colisiones y a la mejora del rendimiento general del filtro. Esta técnica, junto con la correcta elección de los parámetros M, k y n, asegura una operación eficiente y efectiva del Filtro de Bloom en aplicaciones prácticas.

5. Recapitulación y Conclusiones Finales

5.1. Resumen de Actividades

En esta tarea, se implementó y analizó el funcionamiento del Filtro de Bloom, una estructura de datos probabilística utilizada para determinar la pertenencia de un elemento a un conjunto. Las principales actividades incluyeron:

- **Implementación del Filtro de Bloom:** Creación de un arreglo de bits y aplicación de múltiples funciones de hash para insertar y buscar elementos en el filtro.
- **Análisis y Experimentación:** Realización de búsquedas con y sin el uso del Filtro de Bloom, midiendo tiempos de ejecución y tasas de error.
- **Variación de Parámetros:** Investigación de la relación entre el tamaño del arreglo de bits, el número de funciones de hash y la probabilidad de error del filtro, considerando múltiples tamaños de datos.

5.2. Evaluación de Resultados

Los resultados experimentales mostraron que el Filtro de Bloom es efectivo para mejorar los tiempos de búsqueda en bases de datos grandes. Se observaron las siguientes conclusiones:

- **Eficiencia en Tiempos de Búsqueda:** El uso del Filtro de Bloom redujo significativamente los tiempos de búsqueda en comparación con las búsquedas sin filtro.
- **Tasa de Falsos Positivos:** La tasa de falsos positivos fue mínima y dentro de los límites teóricos esperados. Aumentar el tamaño del arreglo de bits y el número de funciones de hash redujo esta tasa, aunque a costa de mayor uso de memoria y tiempo de procesamiento.
- **Equilibrio de Parámetros:** Encontrar un equilibrio adecuado entre el tamaño del arreglo de bits (M), el número de funciones de hash (k) y el número de elementos insertados (n) es crucial para maximizar la eficiencia del filtro.

5.3. Verificación de Hipótesis

Por lo visto los resultados y la discusion asociada, el Filtro de Bloom demuestra ser una herramienta eficiente y efectiva para realizar consultas de pertenencia en grandes bases de datos, validando tanto la teoría como las hipótesis iniciales. Nuestra hipótesis fue que el uso del Filtro de Bloom mejoraría significativamente los tiempos de búsqueda en la base de datos, especialmente con grandes volúmenes de datos, lo cual fue confirmado ya que las búsquedas con el filtro resultaron más rápidas y eficientes. La experimentación corroboró la teoría subyacente del Filtro de Bloom, mostrando cómo la relación entre los parámetros M, k y la probabilidad de error influye en el rendimiento del filtro. Estos hallazgos subrayan la importancia de ajustar adecuadamente los parámetros del filtro para optimizar su desempeño en aplicaciones prácticas.

5.4. Importancia de los Números Primos y Hashing Universal

Durante las pruebas, se encontró que la elección de números primos es esencial para el funcionamiento óptimo del Filtro de Bloom. Los números primos empleados en las funciones hash deben ser superiores a M. Esto se debe a que los números primos grandes garantizan una mejor distribución de los elementos en el arreglo de bits, disminuyendo la posibilidad de colisiones. Además, el uso de hashing universal mejoró considerablemente la distribución uniforme de los bits activados en el arreglo, asegurando que el comportamiento del filtro sea independiente de los datos insertados y aumentando la robustez y eficiencia del filtro.

5.5. Oportunidades de Mejora

A pesar de los resultados positivos, existen varias oportunidades de mejora para el Filtro de Bloom:

- **Optimización de Funciones de Hash:** Investigar y aplicar técnicas más avanzadas para la generación de funciones de hash que puedan mejorar aún más la distribución uniforme y la independencia.
- **Adaptación Dinámica:** Desarrollar un Filtro de Bloom adaptativo que ajuste automáticamente el tamaño del arreglo de bits y el número de funciones de hash en función de la carga de trabajo y el tamaño de los datos.
- **Reducción de Memoria:** Implementar técnicas de compresión para reducir el uso de memoria sin comprometer significativamente la tasa de falsos positivos.
- **Paralelización:** Explorar la posibilidad de paralelizar las operaciones de inserción y búsqueda para mejorar el rendimiento en sistemas con múltiples núcleos.

6. Anexo:

- Página para encontrar números primos http://compoasso.free.fr/primelistweb/page/prime/liste_online_en.php