Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ingeniería en Ciencias de la Computación
CC4102 Diseño y Análisis de Algoritmos

Filtro de Bloom

Tarea 3

Integrantes: Amaro Zurita
Javier Andrés Facondi
Rodolfo Salgado
Profesores: Benjamín Bustos
Auxiliar: Sergio Rojas H.

1. Introducción

Un Filtro de Bloom es una estructura de datos que permite saber rápidamente si un elemento pertenece o no a una base de datos sin tener que realizar una búsqueda exhaustiva sobre ella, lo cual podría ser costoso en términos de tiempo. Debido a la naturaleza probabilística de esta estructura, solo se puede asegurar la no presencia de un elemento, ya que esta puede arrojar falsos positivos con una cierta probabilidad. Resultados que son falsos negativos, es decir que afirme erróneamente que un elemento no esta presente, no son posibles en este método, por lo que estamos ante un algoritmo capaz de generar errores de tipo "one-sided".

En este informe se implementará un Filtro de Bloom en el lenguaje C++. A partir de un dataset dado, se realizará una serie de búsquedas sobre el usando el filtro y luego sin el filtro. Se medirán los tiempos de ejecución y el porcentaje de error asociado al uso del filtro. A partir de estos resultados se evaluará que tan eficaz es implementar un Filtro de Bloom cuando se tienen grandes cantidades de datos a buscar. También se contará la cantidad de errores que se generaran al aplicar el filtro y se determinará como este cambia a medida que se modifican los parámetros y el tamaño del dataset a buscar.

2. Desarrollo

Para esta tarea se implementó en C++ las siguientes funciones y clases:

2.1. Clases implementadas

- **HashFunction:** Representa las funciones Hash que se usarán en el filtro. En este caso se utilizó la función de hash de la librería estándar a la cual se le aplicó el operador módulo para reducir el rango de retorno.
- **BloomFilter:** Dentro de esta clase se implementa el filtro de Bloom. Este mantiene un vector en donde se van almacenando las k funciones de Hash y el vector de m bits. Esta clase posee el método search() que realiza la búsqueda de un string dentro del filtro y retorna si el elemento está o no.

2.2. Funciones implementadas

- **search():** Realiza una búsqueda dentro de los datos usando el método find() de la librería estándar y retorna True si se encontró el elemento.
- **busquedaSinFiltro():** Toma los elementos de un vector de strings y realiza una búsqueda sin aplicar el filtro de Bloom.
- **busquedaConFiltro():** Toma los elementos de un vector de strings y realiza una búsqueda aplicando el filtro de Bloom antes de usar search().
- **gen_vector_N():** Genera un vector de strings de tamaño N en donde una proporción p de los strings provienen del dataset de nombres de bebé y $1 - p$ de los nombres de películas.
- **load_csv():** Carga los archivos .csv en donde se encuentran los datasets.
- **test():** Realiza pruebas del tiempo de ejecución de realizar una serie de búsquedas en el dataset. Recibe como input el número de strings a buscar y la proporción entre nombres de bebé y películas. Los tiempos promedio de ejecución y el porcentaje de error se van guardando en un archivo de texto.
- **main():** Función principal que se encarga de cargar los datasets y llamar a la función test() para realizar pruebas con distintos parámetros.

2.3. Tests

Para evaluar el rendimiento del Bloom Filter, se realizaron una serie de tests que miden el tiempo de ejecución y las tasas de error (falsos positivos) al buscar elementos en conjuntos de datos con y sin el Bloom Filter.

1. **Generación de Conjuntos de Datos:** Se generan conjuntos de datos aleatorios con tamaños $N = 2^{10}, 2^{12}, 2^{14}, 2^{16}$ y proporciones $P = 0, 0.25, 0.5, 0.75, 1$. Cada conjunto contiene una mezcla de nombres de bebés y nombres de películas.

2. **Medición del Tiempo de Ejecución:**

   - **Sin Bloom Filter:** La búsqueda se realiza iterando a través de la lista de nombres.
   - **Con Bloom Filter:** La búsqueda se filtra primero utilizando el Bloom Filter, seguida de una búsqueda completa en los resultados filtrados.

3. **Cálculo de la Tasa de Error:** La tasa de error (porcentaje de falsos positivos) se calcula comparando los resultados de búsqueda con y sin el Bloom Filter.

3. Resultados

Para la ejecución del código se utilizo un PC con las siguientes especificaciones:

- Procesador:
  - Modelo: I9-12900k
  - Cores: 6
  - Threads: 12
  - Frecuencia: 2,9 GHz a 4,3 GHz
  - Cache N2: 1,5MB
  - Cache N3: 12MB
- Memoria principal (RAM): 16 GB a 2666Mhz
- Memoria secundaria :
  - Modelo: WDC PC SN730 SDBQNTY-256G-1001
  - Capacidad: 239 GB
  - Velocidad de lectura: 140 KB/s
  - Velocidad de escritura: 98,1 KB/s
  - Tamaño de bloque: 4096 bytes
- Sistema operativo: Windows 11
- Compilador: GCC 13.2

A continuación presentaremos los gráficos de los resultados obtenidos al realizar los experimentos, luego se analizara y discutirá acerca del comportamiento y rendimiento individual de los métodos de construcción implementados usando el método de búsqueda.

Podemos notar que el tiempo (s) en el caso del filtro aumenta directamente con el porcentaje de palabras que efectivamente se encuentran en el csv.

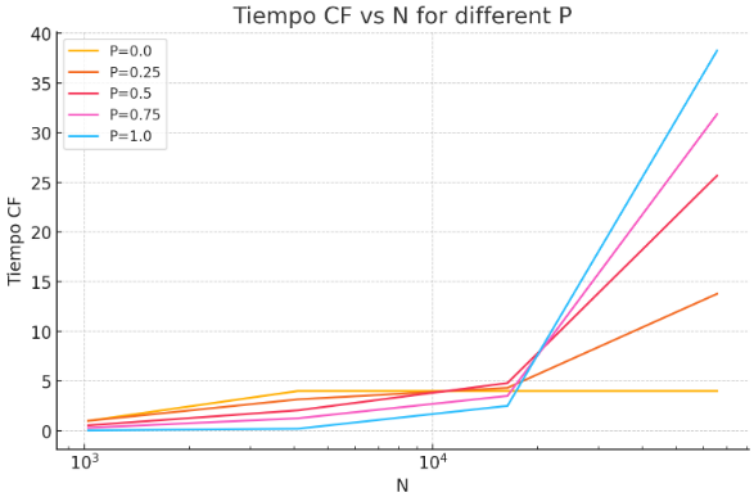

Figura 3.1: Tiempo con filtro para los diferentes P

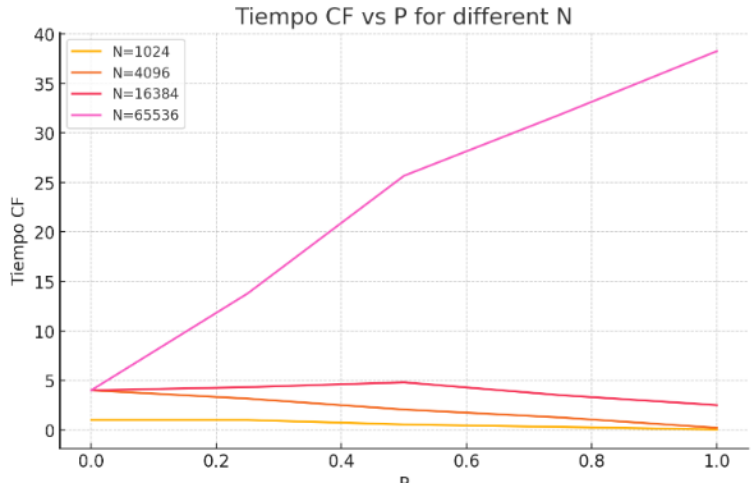

Figura 3.2: Tiempo con filtro para diferente N

Podemos notar que el tiempo (s) en el caso sin filtro aumenta directamente con el porcentaje de palabras que efectivamente se encuentran en el csv al igual que en el caso anterior.

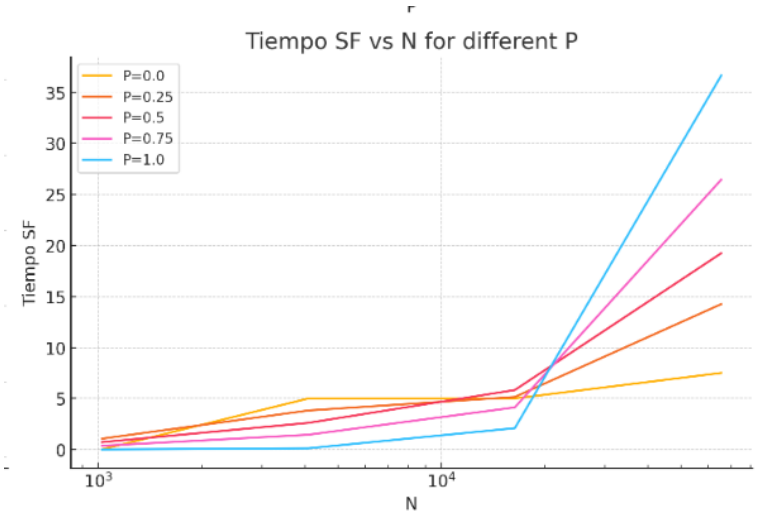

Figura 3.3: Tiempo sin filtro para los diferentes P

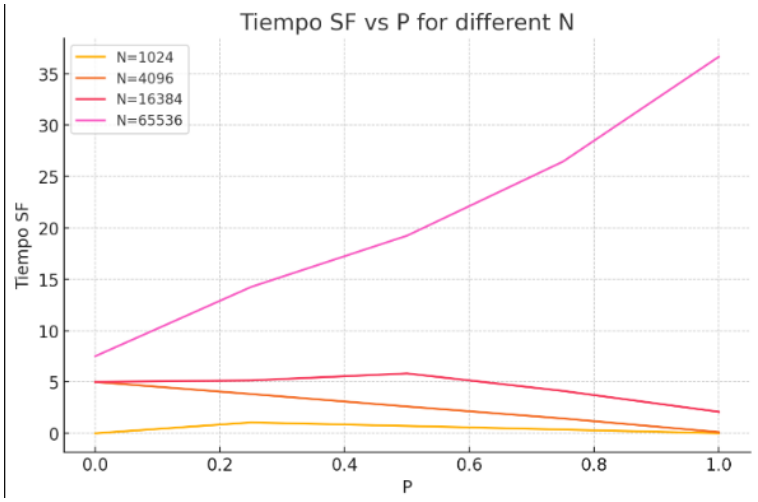

Figura 3.4: Tiempo sin filtro para diferente N

Podemos notar que mientras aumenta el N el porcentaje de error se ve reducido para todos los casos, menos P=1.0, ya que no existe error en ese caso. A su vez podemos notar que

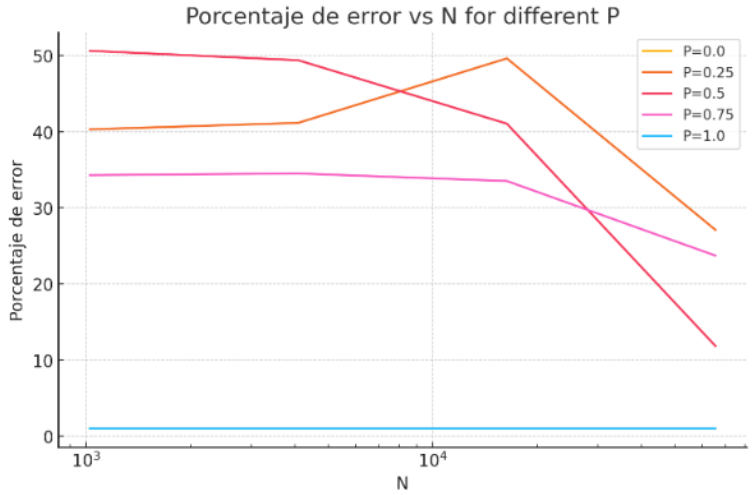

Figura 3.5: Error para diferente P

el porcentaje de error mantiene un comportamiento similar, menos en el caso que nuestro N es igual a 2^{16}

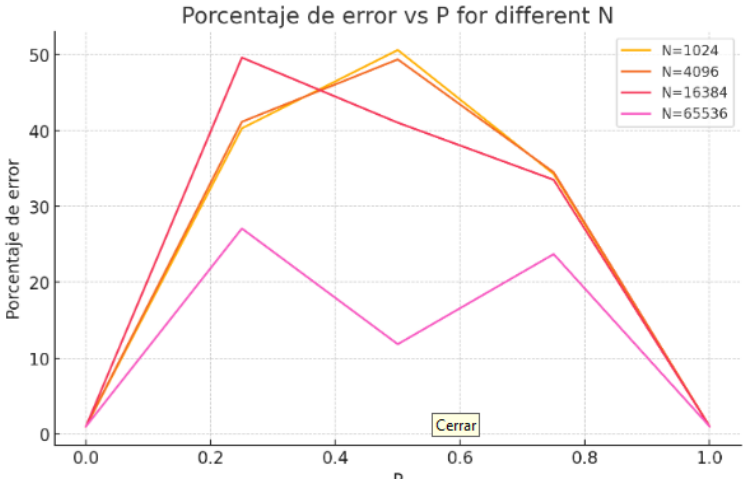

Figura 3.6: Error para diferente N

A continuación se compara para distintos valores de N y P, las búsquedas con el Filtro de Bloom y sin este. Se puede apreciar que sin el filtro la búsqueda tarda más.

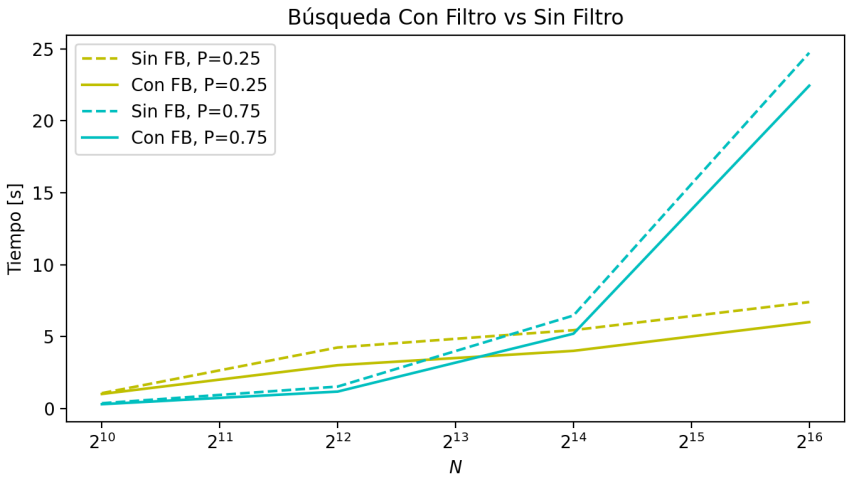

Figura 3.7: Comparación entre búsquedas con Filtro de Bloom y sin Filtro de Bloom

4. Análisis

Eficiencia del Filtro de Bloom: Los gráficos muestran que el uso del filtro de Bloom reduce significativamente el tiempo de búsqueda en comparación con la búsqueda sin filtro, especialmente para conjuntos de datos más grandes.
Impacto del Tamaño del Conjunto de Datos: Tanto el tiempo de búsqueda como el porcentaje de error varían considerablemente con el tamaño del conjunto de datos (N). A medida que N aumenta, el tiempo de búsqueda aumenta exponencialmente, pero el filtro de Bloom mantiene los tiempos más bajos que la búsqueda sin filtro.
Relación entre Proporción p y Error: El porcentaje de error es más alto para valores intermedios de p, lo que sugiere que el filtro de Bloom es más propenso a errores cuando la proporción de coincidencias es moderada.
Consistencia del Error: El porcentaje de error del filtro de Bloom se mantiene relativamente constante o disminuye ligeramente con el aumento de N, lo que indica una buena consistencia en su rendimiento.

4.1. Análisis de parámetros K y M

Se sabe que la formula de un falso positivo en el filtro Bloom es $(1 - e^{-kn/m})^k$, con parámetros k es el número de funciones hash, n es el número de elementos insertados, y m es el tamaño del filtro en bits.
Se realizo un análisis cambiando los parámetros K y M.

1. Parámetro K: Al variar el parámetro K se obtuvo una mejora en cuanto al tiempo de ejecución y una leve mejora en el porcentaje de error lo que fue previsto por la teoría, sin embargo al aumentar mucho el parámetro los resultados se vieron perjudicados.

2. Parámetro M:Al variar el parámetro M se obtuvieron mejores resultados en ejecución y porcentaje de errores esto lo atribuimos a que hay mas espacio bits disponibles para dispersar el hash.

4.2. Comparación entre usar o no el filtro

Como se puede apreciar en la Figura 3.7, cuando el filtro esta activo las búsquedas son ligeramente más rápidas. Igualmente se observa que el comportamiento del tiempo de ejecución no se ve muy alterado para un mismo valor de P. También pareciera que el ahorro de tiempo que provee el filtro va en aumento a medida que se incrementa N.

5. Conclusión

A partir de los resultados obtenidos de la experimentación se puede determinar que los Filtros de Bloom son una buena herramienta para acortar los tiempos de búsqueda cuando se tiene un dataset de gran tamaño. Esta estructura de datos es bastante compacta y su implementación no es muy compleja.

Sin embargo hay que tener en consideración que este método tiene una probabilidad de error asociada, los falsos positivos que el filtro puede generar hacen que aún se tengan que hacer búsquedas ïnútiles"sobre elementos no existentes en la base de datos. Para manejar esta particularidad es necesario considerar las consecuencias de variar la cantidad de funciones de hashing así como también el tamaño del filtro.

Finalmente, es necesario recalcar que si bien se lograron mejoras en el tiempo de ejecución, en un trabajo con un alcance más amplio se podrían haber explorado otros tipos de optimizaciones los cuales podrían haber hecho reducciones más significativas que los Filtros de Bloom.