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

Tarea 3
Filtro de Bloom

Integrantes: Lucas Llort
Franco Navarro
Joaquín Gonzáles

Profesores: Benjamín Bustos

Auxiliar: Diego Salas
Sergio Rojas H.

Ayudante: Martina Mora
Matías Rivera
Vicente Bórquez Z.

Fecha de entrega: 19 de Julio de 2024

Índice

1. Introducción

En este informe se abordará la implementación del filtro de Bloom y su testeo bajo un conjunto con palabras provenientes de los archivos 'Film-Names.csv' y 'Popular-Baby-Names.csv'. El objetivo principal es analizar y comparar el rendimiento del filtro de Bloom implementado, verificando su efectividad y eficiencia en función de diversos tamaños y características de los conjuntos de datos.

Para garantizar la correcta implementación y rendimiento del filtro de Bloom, se diseñaron pruebas detalladas que cubren aspectos como la correcta generación de los conjuntos de datos y la eficiencia de las funciones de hash. Las funciones de hash utilizadas en la implementación se derivan de la librería estándar, asegurando un enfoque robusto y confiable.

En el cuerpo de este informe se proporcionará una descripción detallada de cómo se implementó el filtro de Bloom, incluyendo el proceso de selección y aplicación de las funciones de hash, así como el procedimiento seguido para llevar a cabo las pruebas experimentales. Finalmente, se presentarán los resultados obtenidos y las conclusiones obtenidas a partir del análisis de los resultados.

Como hipótesis inicial creemos que existe una relación directa entre el tamaño del arreglo de bits, el numero de funciones de hash y la probabilidad de de error y que existe una manera de optimizar estas cantidades dado una base de datos concreta.

2. Desarrollo

Para poder llevar a cabo nuestras implementaciones definimos una serie de funciones y clases para este propósito. Por un lado, para buscar en el archivo Popular-Baby-Names.csv ocupando un algoritmo grep se definió la clase grepSearch que se inicializa con un vector llamada words vació y tiene los siguientes métodos.

insert: Este método añade el string entregado al vector de la estructura grepSearch

search: Se busca si existe el string entregado en el archivo y se retorna false en caso de que no este el string y true en caso de que si este.

Por otro lado, para implementar de manera correcta el filtro bloom se decidió que definir una clase llamada "HashFunction" la cual cuenta con el método "operator" el cual recibe un string y retorna un valor que vendría a ser la casilla de M que se volverá true en la ejecución.

En cuanto a la forma de aplicar el filtro bloom, creamos una clase llamada BloomFilter que se inicializa con el arreglo bits con m casillas y cada casilla esta en false. Ademas, esta estructura tambien se inicializa con la variable funcs que es un arreglo con k casillas donde k es un numero entero especificado en la linealización y este arreglo cuenta con k funciones de hashings aleatorias y diferentes entre si. En cuanto los metodos de esta clase, estos son:

insert: Se realiza una iteracion de 0 a k y en cada iteracion se revisa la casilla dada por la funcion de hashing i-esima cuando se le da como argumento el string entregado al metodo y una vez dado el mapeo, se coloca en 1 la casilla de bits.

search: Se procede a buscar un string dado a la funcion iterando desde 0 a k y se va corroborando en cada iteracin si bits(hash(string))==false. En dicho caso, se retorna false y si al finalizar la iteracin nunca se da este caso, se retorna true.

Tambien se definió funciones auxiliares para poder realizar ciertas partes de la ejecucion de manera altamanente coshesionado y poco acoplado. A continuacion, se explican estas funciones.

readFile: Recibe como argumento un string y se intenta abrir el archivo ocupando la ruta especificada en el string en modo lectura. En caso de abrirse exitosamente, se van leyendo linea por linea y se van dejando en un vector que posteriormente es retornado cuando ya no hay mas lineas que leer.

Variables:
- m: Tamaño del arreglo de bits
- k: Cantidad de funciones de hash
- n: Cantidad de palabras que se han ingresado

Tal y como se indico en el enunciado se realizo una investigación para poder desarrollar

Funciones:

- Probabilidad de que un bit no sea elegido por un hash: $1 - \frac{1}{m}$
- Probabilidad de que un bit no sea elegido por ninguno de los k hashes: $\left(\frac{m-1}{m}\right)^k$
- Probabilidad de que un bit sea 0 luego de insertar m palabras: $\left(\frac{m-1}{m}\right)^{kn}$
- Probabilidad de que un bit sea 1 luego de insertar m palabras: $1 - \left(\frac{m-1}{m}\right)^{kn}$
- Probabilidad de un falso positivo: $\left(1 - \left(\frac{m-1}{m}\right)^{kn}\right)^k$

Para que el análisis anterior explicado las funciones de hashing deben ser independiente y uniformemente distribuida, es por esto que usaremos hashing universal:

- Hashing de int: $h_{\text{int}}(x) = ((ax + b) \bmod p)$
  Con a y b números aleatorios entre $[1, p - 1]$, p un número primo muy grande (usaremos el máximo soportado por long long) y x el int

Para manejar un string, se ocupa el hash de la librería estándar y con esto se utiliza la fórmula mencionada.

Derivando la funcion de probabilidad de falsos positivos, y haciendo ciertas simplificaciones, se puede obtener la siguiente formula: $k = \frac{m}{n} \ln 2$, la cual permite calcular el numero optimo de funciones de *hash* que minimiza la probabilidad de falsos positivos del filtro de Bloom dada una cantidad de bits y de palabras ingresadas.

Cantidad de bits	**n° optimo de funciones de hashing**
100000	1
400000	3
900000	7
1500000	11

Tabla 1: Tabla que muestra el número optimo de funciones de hashing para ciertas cantidades de bits, considerando 93890 palabras ingresadas

3. Resultados

n° de bits	n° de funciones de hash	n° de falsos positivos
100000	1	2592
100000	6	3678
100000	11	3773
100000	16	3781
100000	21	3783
100000	26	3783
400000	3	1025
400000	8	1271
400000	13	1987
400000	18	2760
400000	23	3312
400000	28	3551
900000	2	427
900000	3	435
900000	4	432
900000	5	420
900000	6	407
900000	7	406
900000	12	410
900000	17	479
900000	22	613
1500000	2	268
1500000	4	247
1500000	6	239
1500000	8	239
1500000	10	240
1500000	11	238
1500000	16	240
1500000	21	239
1500000	26	248
1500000	31	258
1500000	36	282

Tabla 2: Resultados de experimentación varaindo el número de bits y de funciones de hashing.

Se puede observar que el óptimo para cada conjunto de numero de bits es distinto por lo que en un inicio, no se podría detectar algún patrón que sirva para concluir algo. Los óptimos teóricos para los valores utilizados en la tabla se pueden encontrar en la Tabla 1.

N° total de palabras insertadas	N° de nombres de bebes insertada	tiempo de busqueda de bloom	tiempo de busqueda de grep	N° positivos con bloom	N° positivos busqueda grep
1024	0	0.522105	76.5568	432	25
1024	256	0.45833	50.2909	576	269
1024	512	0.447739	30.7779	725	516
1024	768	0.259322	15.4028	878	768
1024	1024	0.18415	0.514703	1024	1024
4096	0	1.84442	252.893	1734	39
4096	1024	1.62043	181.069	2326	1038
4096	2048	1.39919	129.708	2912	2062
4096	3072	1.08974	70.2583	3507	3086
4096	4096	0.736237	10.3328	4096	4096
16384	0	7.9404	1003.52	6933	125
16384	4096	6.43392	752.895	9295	4182
16384	8192	5.40859	550.248	11655	8242
16384	12288	4.15628	386.361	14016	12313
16384	16384	3.72833	271.47	16384	16384
65536	0	31.756	3996.58	27731	450
65536	16384	26.6569	3222.85	37182	16709
65536	32768	26.9693	3176.25	46621	32993
65536	49152	29.5717	3726.62	56099	49277
65536	65536	34.3333	4950.95	65536	65536

Tabla 3: Resultados del experimento especificado en el enunciado, utilizando 500000 bits y 15 funciones de hashing.

Se puede observar que las búsquedas con grep son totalmente precisas mientras que las búsquedas con filtro Bloon tienen falsos positivos. Por otro lado, los tiempos de búsqueda de los filtros Bloon son mejores que los de grep para cada fila de la tabla

La ejecución de la tarea se llevo a cabo en un computador de estas especificaciones:

- Sistema operativo: Debian 12
- Procesador: Intel Core 3-1115G4

Memoria caché del procesador:
1. L1d: 96 MB (2 instancias)
2. L1i: 64 KB (2 instancias)
3. L2: 2.5 MB (2 instancias)
4. L3: 6 MB (1 instancia)

Memoria RAM: 4 GB

4. Análisis

Se tiene que los resultados obtenidos en la Tabla 2 se ajustan al aspecto teórico investigado puesto que los óptimos observados en la tabla son los mismos que los óptimos obtenidos de manera teórica.

Por otro lado, se observa en la segunda tabla que para todas las filas los tiempos de búsqueda de los filtros Bloom son mejores que los tiempos de búsqueda ocupando *grep* lo cual tiene sentido debido a que el costo del algoritmo filtros Bloom es $\mathcal{O}(k)$ mientras que para *grep* el costo es de $\mathcal{O}(n)$.

No obstante, se sacrifica precisión a la hora de dar verdaderos positivos con el algoritmo de filtros Bloom pero esto no es tan importante ya que este algoritmo se ocupa para extraer filas de bases de datos y, en caso de dar un falso positivo, lo peor que puede pasar es tener que hacer una búsqueda infructuosa dentro de la base de datos a costo $\mathcal{O}(n)$ lo cual hace que este algoritmo demore $\mathcal{O}(k + n) = \mathcal{O}(n)$ dado que k es una constante.

5. Conclusión

A modo de conclusión, el filtro Bloon es una mejor opción que buscar de manera secuencial ocupando un algoritmo de tipo *grep* dado que tiene una cota de peor caso de $\mathcal{O}(k)$ siendo k la cantidad de funciones de *hashings* mientras que un algoritmo de tipo *grep* tiene un costo de peor caso de $\mathcal{O}(n)$ siendo n el numero de filas en el archivo csv.

El análisis anterior se hace teniendo en cuenta que los procesos de búsqueda solo se limitan a aseverar si está o no un elemento en la base de datos mas no el proceso de extracción de la fila que contenga la llave en la base de datos.

Por otro lado, se tiene que la hipótesis planteada inicialmente resulto ser cierta puesto que en nuestra investigación logramos determinar que existen valores óptimos tanto para k como para m siendo k la cantidad de funciones de *hashing* y m el tamaño del arreglo de bits.

En retrospectiva, se alcanzaron los principales objetivos propuestos al inicio de la tarea y se logro comprender la eficacia que pueden llegar a tener los algoritmos probabilísticos por sobre los algoritmos determinísticos en termino de tiempo de procesamiento y por qué en ocasiones sería mas preferible este tipo de algoritmos.