Tarea #3 - Filtros de Bloom

6 de diciembre de 2020 - Gonzalo Navarro & Bernardo Subercaseaux

Como ya se exploró en la Tarea 1, las lecturas y escrituras en disco son varios órdenes de magnitud más lentas que su contraparte en RAM. Una consecuencia natural de este hecho es que conviene evitar acceder al disco en la medida de lo posible. Consideremos un archivo en disco que mantiene una lista de palabras L, y frecuentemente se desea revisar si una palabra p está o no en L. Una forma clásica de disminuir los accesos a disco consiste en utilizar una *caché* que guarda en RAM las palabras p más frecuentemente consultadas.

En esta tarea estudiarán los *filtros de Bloom*, una idea complementaria a las cachés. Un filtro de Bloom B es una estructura de datos que se mantiene en RAM y que procesa cada consulta por una palabra p resultando en un valor $B(p)$ que puede ser 0 o 1. Si $B(p) = 1$, la palabra p *quizás* esté en L, mientras que si $B(p) = 0$, la palabra p *definitivamente no* está en L. Esto permite por supuesto evitar buscar en disco cualquier palabra p tal que $B(p) = 0$.

1. Filtros de Bloom

Un filtro de Bloom B está determinado por su tamaño m y k funciones de hash $h_1, \ldots, h_k$ que retornan siempre valores en $[0, m-1]$. Se mantiene un vector de bits V de tamaño m, donde inicialmente cada posición parte en 0. Los filtros de Bloom soportan dos operaciones:

- **`Insertar`**(p): Por cada función de hash h_i, se hace $V[h_i(p)] = 1$.
- **`Revisar`**(p): Si para alguna función de hash h_i se cumple que $V[h_i(p)] = 0$ entonces se retorna 0, de lo contrario se retorna 1.

Por ejemplo, tomemos $m = 5, k = 2, h_1(x) = 6x \mod 5$ y $h_2(x) = x^2 + 2 \mod 5$. Consideremos la siguiente secuencia de operaciones (Figura 1):

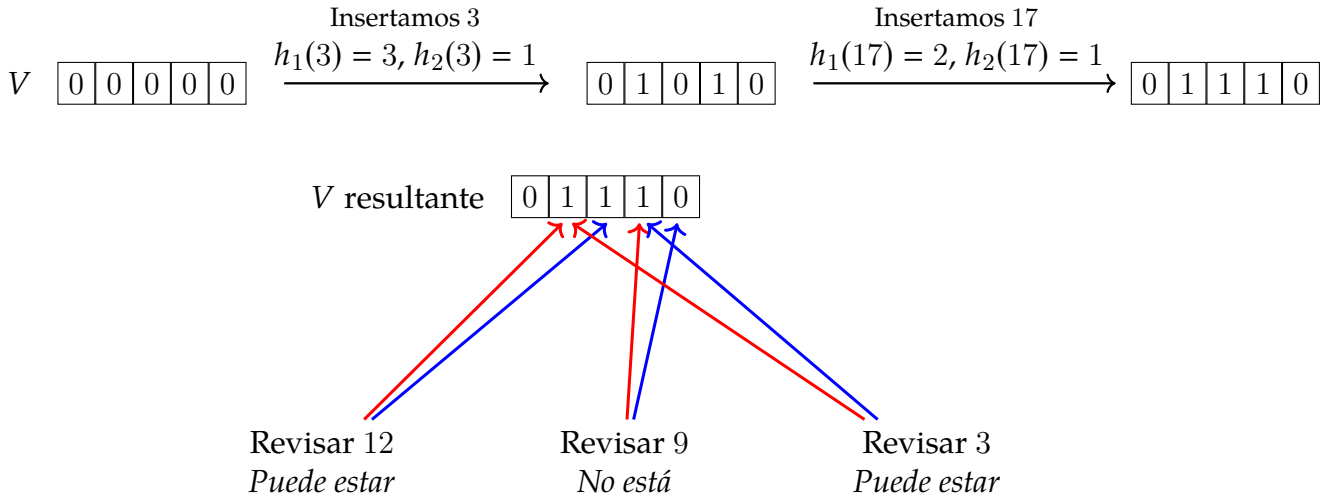

Figura 1: Ejemplo de operaciones sobre un filtro de Bloom. El resultado de h_1 está representado por una flecha azul, mientras que el de h_2 está representado por una flecha roja.

1. (0.2 pts.) Pruebe que efectivamente los filtros de Bloom cumplen que (i) si `Revisar` retorna 0 sobre un elemento, entonces este jamás se ha insertado y (ii) si un elemento se ha insertado en el filtro, entonces `Revisar` retorna 1 sobre él.

2. (0.5 pts.) Implemente en Python (≥ 3.6) un filtro de Bloom. En su construcción debe recibir como parámetros los valores m y k. Como función de hash se sugiere el uso de Murmur. Para obtener k funciones de hash distintas puede guardar k semillas generadas aleatoriamente, y utilizar la función de *Murmur* parametrizada por tales semillas. Es decir, si s_i es la i-ésima semilla aleatoria, puede usar $h_i(x) := \texttt{Murmur}(x, s_i)$. Puede asumir que los elementos a insertar serán strings. Además, para guardar el vector V de manera eficiente en espacio se recomienda usar un vector de bits.

En el ejemplo de la Figura 1 se muestra que los filtros de Bloom a veces dicen que un elemento p (12 en el ejemplo) *puede* haber sido insertado aun siendo que tal elemento jamás se insertó. Diremos que tales resultados son *falsos positivos*. Los siguientes apartados buscan acotar la probabilidad de obtener falsos positivos. Asuma que las k funciones de hash son totalmente independientes, y que retornan valores entre 0 y $m-1$ de manera aleatoria y con distribución uniforme.

3. (0.2 pts.) Para todo bit i de V, se define el evento E_i como *que ese bit esté en* 1 *tras realizar* n *inserciones*. Pruebe que

$$Pr(E_i) = 1 - \left(1 - \frac{1}{m}\right)^{kn} \approx 1 - e^{-kn/m}$$

donde la última aproximación está tomada cuando m tiende a infinito.

4. (0.3 pts.) Asuma por un momento que los eventos E_i son todos independientes. ¿Cómo calcularía la probabilidad de obtener falsos positivos en este caso? Argumente que los eventos E_i y E_j no son independientes si $i \neq j$.

5. (0.3 pts.) Sea X la variable aleatoria que representa la cantidad de bits en 1 tras n inserciones distintas. Utilice el apartado 3 para deducir que

$$E(X) = m\left[1 - \left(1 - \frac{1}{m}\right)^{kn}\right]$$

y argumente también que si denotamos con FP la probabilidad de un *falso positivo*, entonces se cumple que

$$Pr(\text{FP} \mid X = x) = \left(\frac{x}{m}\right)^k$$

6. (0.4 pts.) Suponga (sin fundamento) que la variable X está muy *concentrada* con respecto a su esperanza, es decir, que $Pr(X = x)$ vale 1 si $x = \lfloor (E(X)) \rceil$ (el entero más cercano a $E(X)$) y 0 si no. Pruebe que bajo esta suposición se cumple que

$$Pr(\text{FP}) = \sum_{x=1}^{m} Pr(\text{FP} \mid X = x) Pr(X = x) \approx \left[1 - \left(1 - \frac{1}{m}\right)^{kn}\right]^k$$

Note que es relevante el signo de aproximación en lugar de igualdad, ya que $E(X)$ no es necesariamente entero.

7. (0.3 pts.) Pruebe que la probabilidad de falsos positivos se minimiza tomando $k = \frac{m \ln 2}{n}$.

8. (0.3 pts.) Elabore un gráfico que permita entender mejor la expresión del lado derecho del apartado anterior. Para ello considere $k = \lceil \frac{m \ln 2}{n} \rceil$. ¿Es baja la probabilidad si se han hecho muchas inserciones (p. ej. $n \geq 5m$)? ¿Qué tal si se han realizado muy pocas (p. ej. $m \geq 5n$)? Utilice en su informe un ejemplo concreto de valores de m y n que ilustre la efectividad de los filtros de Bloom.

2. Filtros de Bloom para mejorar la latencia de una aplicación

En la sección anterior se analizaron los filtros de Bloom desde un punto de vista teórico, mostrando que sirven para evitar buscar elementos que nunca se han insertado a una lista que se mantiene en disco. Esta sección trata el problema práctico asociado en mayor detalle.

Imagine que desea guardar n pares *(nombre de usuario, e-mail)* en disco. A veces, con el objetivo de recuperar su contraseña, los usuarios ingresan su nombre de usuario, y reciben un correo con instrucciones para recuperar su contraseña. Sin embargo, muchas veces ocurre que usuarios que nunca se han registrado intentan recuperar una contraseña, y por supuesto sus nombres de usuario se buscan infructuosamente en el disco, lo que desperdicia tiempo de computo.

9. (0.3 pts.) Programe un módulo que recibe un entero n, inserta n strings aleatorios (representando nombres de usuario) a un filtro de Bloom B, e inserta también esos n strings, acompañados de e-mails aleatorios, a un archivo en disco L. Como formato para el archivo utilice una línea por cada par de datos, separando ambos elementos con un espacio (<username> <email>). Por ejemplo, un archivo puede verse de la siguiente forma:

```
gN4v4rr0 olaznog@wow.owo
bernieSubs odranreb@uwu.owo
jayPerez egroj@owo.uwu
...
```

Debe programar un módulo que busca en el archivo L las palabras que pasan el filtro. Para buscar en el archivo utilize la herramienta grep. El siguiente extracto ejemplifica su uso:

```
import os
username = 'jayPerez'
cmd = f"grep '{username}' L.txt"
os.system(cmd)
```

10. (0.3 pts.) Programe un módulo, que usará para experimentar, que utiliza el módulo anterior, y luego procesa Q consultas generadas al azar[1]. Para cada nombre de usuario q consultado, se revisa primero si el filtro de Bloom dice que puede estar en L o no. Si el filtro de Bloom dice que q no está en L, entonces se responde eso al usuario sin errores. Si dice que puede estar, entonces se procede a buscar en L.

[1]En la sección 3 se discute este proceso de generación.

3. Experimentación

La idea detrás del uso de los filtros de Bloom es evitar accesos innecesarios al disco. En teoría eso debería reducir el tiempo asociado al total de una serie de consultas. Sin embargo, para que los filtros de Bloom sean realmente efectivos, se suele requerir que el valor de m (y por tanto el espacio asociado) sea grande. Su informe debe plantear claramente este *trade-off*, realizar hipótesis al respecto y finalmente intentar validarlas experimentalmente.

3.1. Diseño experimental

Para que los experimentos permitan una buena aproximación a la realidad, debe ocurrir tanto que los usuarios consultan con nombres de usuario que efectivamente están en L como que no. Eso quiere decir que la generación de consultas aleatorias debe tener en cuenta esto.

11. (0.5 pts.) Diseñe una forma de generar las consultas que cumpla con lo anterior. Hay muchas opciones posibles y válidas, lo importante es que justifique su metodología.

3.2. Hipótesis

12. (0.4 pts.) Describa sus hipótesis con respecto a la eficiencia temporal y espacial de los filtros de Bloom en relación a los parámetros previamente definidos (m, n, k). **Explique el razonamiento detrás de sus hipótesis**. No es necesario que las hipótesis sean correctas, solo que sean serias y honestas.

3.3. Experimentos

13. (0.6 pts.) Describa valores de n, m y k con los cuáles experimentará. Experimente también sobre la probabilidad de falsos positivos, y estudie si se condice con lo analizado teóricamente. Considere usar valores de n suficientemente grandes para que el tiempo de una búsqueda en disco sea significativo (p. ej. en el rango de 20-500 milisegundos). Experimente tanto en tiempo utilizado como en espacio. Este post puede ser útil para medir el espacio usado por su programa.

4. Resultados

14. (0.7 pts.) Grafique claramente los resultados de sus experimentos. No debería incluir más de 6 gráficos (seleccione los que mejor manifiestan las ideas que quiere comunicar) en el cuerpo de su informe, pero puede incluir gráficos menos importantes en el anexo.

15. (0.3 pts.) Contraste los resultados obtenidos con las hipótesis planteadas.

5. Conclusión

16. (0.4 pts.) Tras contrastar los resultados con las hipótesis planteadas, concluya sobre los resultados obtenidos. Una conclusión no consiste en repetir lo ocurrido, sino en darle sentido y perspectiva. Idealmente la conclusión hace sentido a su lector *después* de haber leído el resto del documento.