Bloom Filter

Jhonny Lorenzo

June 25, 2018

Abstract

Bloom Filter, es una estructura de datos compacta que nos informa si un elemento está presente en un conjunto determinado. Estudiaremos el porqué de su rendimiento frente a otras estructuras de datos, su sustento matemático, una gráfica de los parámetros de esta estructura y conclusiones.

1 Introduccion

Bloom Filters es una estructura de datos compacta para representar probabilísticamente un conjunto para admitir consultas de la presencia de elementos sobre un conjunto, es decir, ¿El elemento X se encuentra en el conjunto Y?.

Esta representación compacta es la recompensa por admitir una pequeña tasa de falsos negativos en las consultas de existencia, es decir, las consultas pueden reconocer incorrectamente un elemento como si se encontrara en el conjunto. Es por esto que, el Bloom Filter se emplea para comprobar si un elemento no está presente, ya que no garantiza falsos negativos. Esto le permite no realizar trabajos adicionales para elementos que no existen en un conjunto.

2 Teoría

2.1 Falso positivo

Es un resultado que indica que existe una condición dada, cuando en realidad no existe. Es decir, es un error en el que la prueba verifica una sola condición y erróneamente da una decisión afirmativa.

Ejm. *Cuando un Software Antivirus detecta un archivo normal como si fuera un virus.* La condición es *el archivo normal es un virus*, cuando en realidad es *el archivo no es un virus*, se comete un error positivo, pues así se garantiza que el computador este libre de virus, a pesar de que dicho archivo no lo fuese.

2.2 Falso negativo

Es un resultado de prueba que indica que una condición no se cumple, mientras que de hecho si se cumple. En otras palabras, erróneamente no se ha inferido ningún efecto.

Ejm. Una prueba que indica que una mujer no está embarazada mientras que ella realmente lo está.

2.3 Bloom Filter

Un Bloom Filter consiste de un vector A de tamaño n, A[0] a A[n-1], inicializados en 0. Un Bloom Filter usa k funciones hash $h_1, h_2, ..., h_k$ con un rango de {0,...,n-1}.
Una vez definido esto, comencemos con el análisis matemático.

3 Bloom Filter: Sustento matemático

Consideremos un conjunto $A = \{a_1, a_2, ..., a_n\}$ de n elementos. Bloom filters describe información de pertenencia de A utilizando un vector V de tamaño m.

Para esto, tenemos k funciones hash, $h_1, h_2, ..., h_k$ con $h_i : X \to \{1..m\}$, se usan como se describe a continuación

Observemos que después de ingresar n claves en un filtro de tamaño m usando k funciones hash, la probabilidad de que un bit en particular a?n sea 0 es:

$$p_0 = (1 - \frac{1}{m})^{kn} \approx e^{-\frac{kn}{m}} \ ... \ (1)$$

Tenga en cuenta que suponemos funciones hash perfectas que separan los elementos de A uniformemente en todo el espacio $\{1..m\}$.

Por lo tanto, la probabilidad de un falso positivo (la probabilidad de que todos los k bits se hayan establecido previamente) es:

$$p_{err} = (1 - p_0)^k = (1 - (1 - \frac{1}{m})^{kn})^k \approx (1 - e^{-\frac{kn}{m}})^k = (1 - p_0)^k \ ... \ (2)$$

De ahora en adelante. por conveniencia usamos las aproximaciones asint?ticas p_{err} y p_0 para representar (respectivamente) la probabilidad de que un bit en el Bloom Filter sea 0 y la probabilidad de un falso positivo.

Supongamos que se nos da m y n y deseamos optimizar el número de funciones hash k para minimizar la probabilidad del falso positivo p_{err}.

El número óptimo de funciones hash que minimiza f en función de k se encuentra fácilmente tomando la derivada. Entonces:

Sea $g(k) = kln(1 - e^{-\frac{kn}{m}})$, donde $p_{err} = e^{g(k)}$ y minimizar el falso positivo de p_{err} es equivalente a minimizar $g(k)$ respecto de k, entonces:
$\frac{dg(k)}{dk} = ln(1 - e^{-\frac{kn}{m}}) + \frac{kn}{m} \frac{e^{-\frac{kn}{m}}}{(1-e^{-\frac{kn}{m}})}$

Realizando operaciones llegaremos a la conclusión que para $k = ln(2)\frac{m}{n}$ es el mínimo global. Por lo tanto, reemplazando este resultado en la ec.1 tendriamos

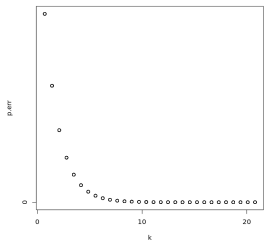

Figure 1: Comportamiento de p.err respecto de k funciones hash

que el p_0 es igual a $\frac{1}{2}$, esto último lo reemplazamos en (2) y obtenemos p_{err} es igual a $(\frac{1}{2})^k \approx (0.6185)^{\frac{m}{n}}$.

Ahora sustituiremos el valor optimo de $k = \frac{m}{n}ln2$ en la ec. 2:

$$p_{err} = (1 - e^{-\frac{kn}{m}})^k = (1 - e^{-\frac{ln(2)\frac{m}{n}n}{m}})^{ln(2)\frac{m}{n}}$$

Se simplica:
$ln(p) = -\frac{m}{n}(ln2)^2$

Operando, encontramos que la cantidad óptima de bits por elemento es:

$$\frac{m}{n} = -\frac{log_2(p)}{ln2} \approx -1.44log_2(p) \ ... \ (3)$$

Reemplazando el valor optimo de $k = \frac{m}{n}ln2$ en la ec. 3, obtenemos lo siguiente:

$$k = -\frac{lnp}{ln2} = -log_2(p)$$

Es decir, que la cantidad de funciones hash del Bloom Filter, solo depende de la probabilidad del falso positivo p_{err}.

Esto significa que para una probabilidad de falso positivo p dada, la longitud de un Bloom Filter m es proporcional al número de elementos que se filtran n, y el número requerido de funciones de dispersión solo depende de la probabilidad de falso positivo $p.err$

4 Resultados Graficos

Se analizo la caida exponencial de $p.err$ respecto de las k funciones hash, veamos el grafico:

5 Conclusión

Al parecer guiandonos de la gráfica tenemos que para un determinado de k numeros de funciones hash, no se aprecia el aumento del cifrado.

References

[1] Mitzenmacher, M., Upfal, E. (2017). Probability and computing: Randomized and probabilistic techniques in algorithms and data analysis. Cambridge; New York ; Port Melbourne ; Delhi ; Singapore: Cambridge University Press.