Dorina Ismaili & Deniz Gaiser
18.11.2024

Mining Massive Datasets
Homework 4

Exercise 1

Consider a cluster of n machines. Each has a probability p of failing in a given period of time T.

a) Probability of no failure is $P(\text{no failure}) = (1-p)^n$. Probability of at least one failure is $P = 1-(1-p)^n$.

b) The case follows a binomial distribution. The probability of exactly k failures out of n machines is

$$p_k = \binom{n}{k} p^k (1-p)^{n-k}$$

.

c) The probability is $\sum_{k=1}^{n} p_k = 1$. For $p_0 = \binom{n}{0} p^0 (1-p)^n = (1-p)^n$.

$$\sum_{k=1}^{n} p_k = \sum_{k=0}^{n} p_k - p_0 = 1-(1-p)^n$$

.

Exercise 2

Give two examples for hash functions as alternatives to the arithmetic remainder function mod.

1. Multiplicative Hashing: Multiplicative hashing uses a combination of multiplication and extraction of specific bits to produce a hash value. The general form is: $h(x) = \lfloor m \cdot (x \cdot A \bmod 1) \rfloor$.
2. DJB2 processes the string by running it through a seed. In this method, the key operation lies in the logical shift and the addition of both the seed and one character of the input. This is then done on every character of the input, with the seed being the previous result respectively.

Hashes are used as a one-directional data store for passwords. The goal of hashing is to create a fixed mapping, which ideally can not be reversed efficiently.
The security breach described in the video happened due to the lack of security measures, with the encryption in itself being sufficient. Naive Hashing also has effects similar to encryption, namely that for a fixed input, you get a fixed result. To avoid this, more countermeasures in addition to hashing have to be made.
Rainbow Tables are used, to store prevalent inputs for certain hashing or encryption algorithms, to reduce computing time, and to crack passwords more efficiently. It is more or less a pre-defined set of inputs for a brute-force attack.
To avoid significant overlaps between similar user passwords, another user-specific factor has to be implemented into the data store. This is called salt, and it includes an user-specific factor into the computed hash.

Exercise 3

- In Java, a HashMap stores key-value pairs in a data structure of buckets, mostly LinkedLists or BinaryTrees. The essential component of the HashMap is the hashCode method, which can be overridden and customized.
- When adding a new entry with put(¡key¿, ¡value¿), at first the hashCOde method is called on the key. If the computed hashCode is unique, a new entry is generated and then happened to the HashMap. If not, the existing value is overridden. If the exceeds a new power of 2, the hashmap is additionally reinitialized with double the size.
- With the get(key) method, the hashCode method is run again toobtain the hashCode of the key. If the HashCode is not in the HashMap, null is returned. Otherwise, a search is performed on the key-value pairs, according to the data structure, i.e., a binary search.

Exercise 4

Suppose you have to design/implement hash functions for a Bloom filter algorithm. Propose 3 hash functions which are as independent from each other as possible and briefly justify your choice.

1. MurmurHash3 is a fast, non-cryptographic hash function known for its high-quality, uniformly distributed outputs. h1(x)=MurmurHash3(x,seed1)
2. DJB2 is a simple hash function that produces a hash value by iteratively combining input bytes with bitwise shifts and additions. Formula: $h(x) = 5381$; $h(x) = ((h << 5) + h) + c$
3. FNV is a fast hash function that combines the input data using XOR and multiplication with a large prime number. For each byte of input b: $h(x) = (h(x) \oplus b) \cdot FNV_{prime}$.

	MurmurHash3	DJB2	SDBM
formula	$h1(x) = MurmurHash3(x, seed1)$	$h(x) = 5381$; $h(x) = ((h << 5) + h) + c$	$h(x) = (h(x) \oplus b) \cdot FNV_{prime}$
focus	bit-rotation and multiplication constants	logical shift multiplication	primary numbers
salt	random seed	random seed (usually 5381)	prime numbers

The choice falls on those three hash functions, due to the focus of their implementations being different. On one hand, we have multiplication with primary numbers, on the other with powers of 2. Additionally one non-multiplication-based approach with logical operators.

Exercise 5

Consider a Bloom filter with a bit array of n = 5 bits.

a) After hashing m elements, the probability that a specific bit remains unset is

$$P(\text{bit not set}) = (1 - \frac{1}{n})^{km}$$

. It holds that

$$P(\text{bit set}) = 1 - (1 - \frac{1}{n})^{km}$$

b) Given hash functions $k = 2$, $h_1(x) = x \mod 5$, $h_2(x) = 2x + 3 \mod 5$. The bit array has 5 bits, indexed as 0,1,2,3,4. Elements to hash: $x = 4$ and $x = 1$. Bit array is [0,0,0,0,0]. For $x = 4$, $h_1(x) = 4$ meaning that it sets bit 4 to 1. $h_2(x) = 1 (\text{mod } 5)$ meaning that it sets bit 1 to 1. So the bit array is [0,1,0,0,1]. After hashing for $x = 1$, $h_1(x) = 1 (\text{mod } 5)$ sets bit 1 to 1. For $h_2(x) = 0 (\text{mod } 5)$ sets bit 0 to 1. The final bit array is $[1, 1, 0, 0, 1]$. Both hash functions are uniformly distributed, meaning all bits are equally likely to be hit.

c) The false positive probability P_{FP} for a Bloom filter

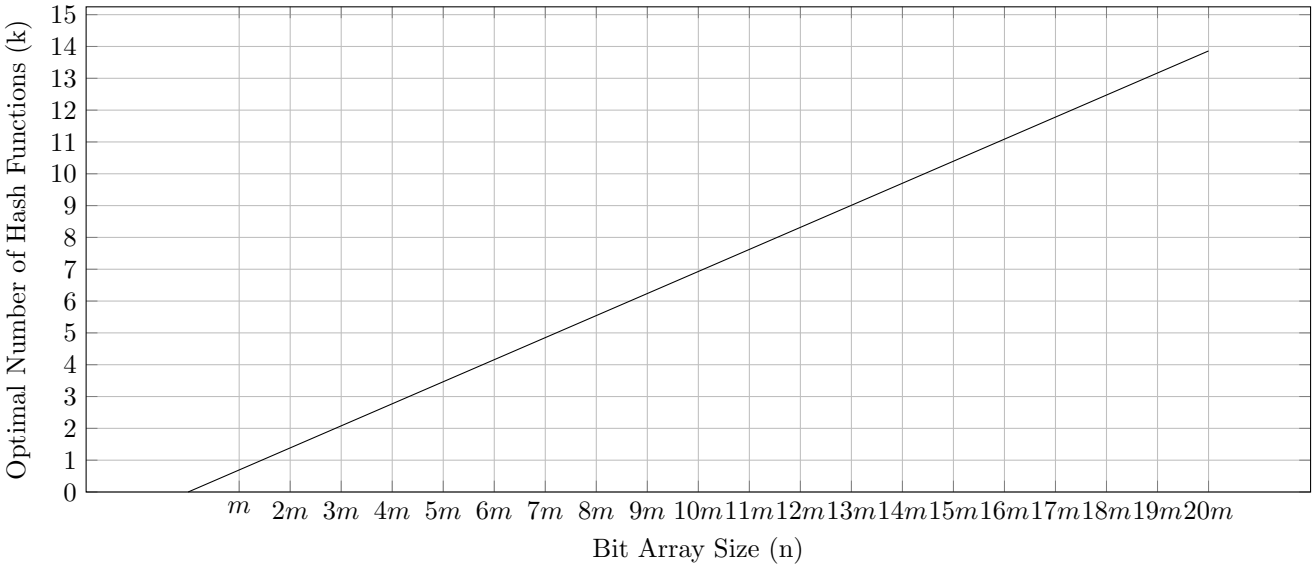
$$(1 - (1 - \frac{1}{n})^{km})^k$$

. Computing the result we have

$$P_{FP} = (1 - (1 - \frac{1}{5})^4)^2 = (1 - \frac{256}{625})^2 = (\frac{369}{625})^2 = 0.3485.$$

The probability is 34%.

Figure 1: Task 6 a)
Optimal Number of Hash Functions (k) for Different Bit Array Sizes (n)



Exercise 6

a) We want to determine the values, so that the false positive rate $P_{false_positive} = (1-(1-\frac{1}{n})^{k \cdot m})^k \approx (1-e^{-k \cdot \frac{n}{m}})^k$ is minimized.
With this, we can determine that the ideal k can be plotted with $k = \frac{n}{m} \cdot \ln(2)$
For the plot, see Figure 1.
It is observable, that k is linear-proportional to n.

b) A short program to compute the results was attached to the zip file in "e04/Task 6b.py"

b	n	k	$P(FPP)$
0.01	9625	7	0.009842603536832032
0.005	11125	8	0.004782467206562176
0.001	14500	10	0.0009429340108142648

We don't have to know m, as m is a scalar, and thus does not change the proportionality.

Exercise 7

Consider again the Bloom filtering technique. Prove that the optimal number k_{opt} of hash functions for given size n of the bitset and a given number m of keys is $k_{opt} = \frac{n}{m} \ln(2)$.

The false positive probability of a Bloom Filter is

$$P_{FP} = (1 - e^{-km/n})^k$$

. Our goal is to minimize P_{FP} with respect to k. Let $x = e^{-km/n}$. We take the logarithm on both sides as we have

$$ln(P_{FP}) = ln(1-x)^k$$

We take the derivative on both sides such that $\frac{d}{dx} ln(P_{FP}) = 0$ It holds that

$$\frac{d}{dk} ln(P_{FP}) = ln(1-x) + k \frac{-1}{1-x} \frac{dk}{dx},$$

where $\frac{dx}{dk} = -\frac{m}{n}x$. Therefore it holds

$$ln(1-x) = \frac{km}{n}\frac{1-x}{x}$$

. Rearranging it holds that $k_{opt} = \frac{n}{m}\ln(2)$.

Exercise 8

a) key attributes: University, Course-Id
With those, we can access each course, and execute queries on them. The students and grades are irrelevant, as the grades don't tell about the course. The students can also be omitted, as we only want the count of students.

b) key attributes: University, Student
Since we only want every student with a grade better than 2.0, we do not consider the courses, as grading is uniform. The other fields are required, as we need to construct unique IDs with university + course. The grade can then be obtained via that ID, which is why we also do not need the grades as keys.

c) key attributes: University, Course-Id
We want to extract the grades from courses to get the average grades. For this, we do not need the students, since we want an average grade. The grades themselves are obtained by looking for the university + course-Id.