# Daisy Bloom Filters

**Ioana O. Bercea**
KTH Royal Institute of Technology, Stockholm, Sweden

**Jakob Bæk Tejs Houen**
BARC, University of Copenhagen, Denmark

**Rasmus Pagh**
BARC, University of Copenhagen, Denmark

## Abstract

A filter is a widely used data structure for storing an approximation of a given set $S$ of elements from some universe $\mathcal{U}$ (a countable set). It represents a superset $S' \supseteq S$ that is "close to $S$" in the sense that for $x \notin S$, the probability that $x \in S'$ is bounded by some $\varepsilon > 0$. The advantage of using a Bloom filter, when some false positives are acceptable, is that the space usage becomes smaller than what is required to store $S$ exactly.

Though filters are well-understood from a worst-case perspective, it is clear that state-of-the-art constructions may not be close to optimal for particular distributions of data and queries. Suppose, for instance, that some elements are in $S$ with probability close to 1. Then it would make sense to always include them in $S'$, saving space by not having to represent these elements in the filter. Questions like this have been raised in the context of Weighted Bloom filters (Bruck, Gao and Jiang, ISIT 2006) and Bloom filter implementations that make use of access to learned components (Vaidya, Knorr, Mitzenmacher, and Krask, ICLR 2021).

In this paper, we present a lower bound for the expected space that such a filter requires. We also show that the lower bound is asymptotically tight by exhibiting a filter construction that executes queries and insertions in worst-case constant time, and has a false positive rate at most $\varepsilon$ with high probability over input sets drawn from a product distribution. We also present a Bloom filter alternative, which we call the *Daisy Bloom filter*, that executes operations faster and uses significantly less space than the standard Bloom filter.

**2012 ACM Subject Classification** Theory of computation → Data structures design and analysis

**Keywords and phrases** Bloom filters, input distribution, learned data structures

**Digital Object Identifier** 10.4230/LIPIcs.SWAT.2024.9

**Related Version** *Full Version*: `https://arxiv.org/abs/2205.14894`

**Funding** Supported by grant 16582, Basic Algorithms Research Copenhagen (BARC), from the VILLUM Foundation.

## 1 Introduction

This paper shows asymptotically matching upper and lower bounds for the space of an optimal (Bloom) filter when the input and queries come from specific distributions. For a set $S$ of keys (the input set), a filter on $S$ with parameter $\varepsilon \in (0,1)$ is a data structure that answers membership queries of the form "is x in S?" with a one-sided error: if $x \in S$, then the filter always answers YES, otherwise it makes a mistake (i.e., a false positive) with probability at most $\varepsilon$. The Bloom filter [9] is the most widely known such filter, although more efficient constructions are known [2,3,5,6,20,25,34,41,42,45]. Filters are also intimately related to dictionaries (or hash tables), the latter of which always answer membership queries exactly.

When errors can be tolerated, (Bloom) filters are much better than dictionaries at encoding the input set: they require $\Theta(n \log(1/\varepsilon))$ bits to represent a set of size $n$, versus the $\geq n \log(u/n)$ bits that a dictionary would require (here, $u$ is the size of the universe). As such, filters are often used in conjunction with dictionaries to speed up negative queries. In particular, filters are often stored in a fast but small memory and are used to "filter out" a majority of negative queries to a dictionary (which might reside in big but slow memory). Because of this, they have proved to be extremely popular in practice and research on them continues to this day, both in the direction of practical implementations [19, 24, 44] and on the theoretical front [4, 5, 34]. For instance, recent advances in filter design have included making them dynamic, resizeable and lowering the overall space that they require.

**The filter encoding.** In this paper, we ask ourselves what should optimal filters look like when they encode sets that come from a specific distribution. While this question has been resolved for exact encodings (i.e., entropy), no similar concepts are known for filter encodings. Indeed, considering input distributions raises several technical questions. For instance, it is not even clear how to define the concept of approximate membership with respect to a set drawn from a distribution. Should we assume that the input set is given to us in full before we build our filter and allocate memory? Moreover, we would like to obtain designs that are never worse than filters with no knowledge of the input distribution, both in space allocated and time required to perform every operation. Should we then require that the false positive guarantee hold for every possible input set or just on average over the input distribution?

We also study optimality when additionally, we have access to a distribution over queries. This is especially important for applications in which the performance of the filter is measured over a sequence of queries, rather than for each query separately [11, 26]. At the extreme end of this one can consider adversarial settings, in which an adversary forces the filter to incur many false positives (which can cause a delay in the system by forcing the filter to repeatedly access the slow dictionary). In these settings, defining what it means for the filter to behave efficiently can be a challenge and several definitions have been considered [2, 38–40]. For us, the challenge is to use the query distribution to obtain gains, while making sure that the filter does not on average exhibit more false positives than usual. This is natural when each false positive has the same cost, independent of the query element.

To this end, we consider a natural generative model of input sets and queries. Specifically, we let $\mathcal{P}$ and $\mathcal{Q}$ denote two distributions over the universe $\mathcal{U}$ of keys and let $p_x$ (and $q_x$, respectively) denote the probability that a specific key $x \in \mathcal{U}$ is sampled from $\mathcal{P}$ (and $\mathcal{Q}$, respectively). The input set $S$ is generated by $n$ independent draws (with replacement) from $\mathcal{P}$ and we let $\mathcal{P}_n$ denote this product distribution.[1]

We then define approximate membership for a fixed set $S$ to mean that the average false positive probability over $\mathcal{Q}$ is at most $\varepsilon$. Specifically, let $\mathcal{F}$ denote the filter and let $\mathcal{F}(S, x) \in \{\mathsf{YES}, \mathsf{NO}\}$ denote the answer that $\mathcal{F}$ returns when queried on an element $x \in \mathcal{U}$, after having been given $S \subseteq \mathcal{U}$ as input. Then we propose the following definition:

▶ **Definition 1.** *For any $\varepsilon$ with $0 < \varepsilon < 1$, we say that $\mathcal{F}$ is a $(\mathcal{Q}, \varepsilon)$*-filter for *$S$ if it satisfies the following conditions:*

1. *No false negatives: For all $x \in S$, we have that $\Pr[\mathcal{F}(S, x) = \mathsf{YES}] = 1$.*
2. *Bounded false positive rate:*

$$\sum_{x \in \mathcal{U} \setminus S} q_x \cdot \Pr[\mathcal{F}(S, x) = \mathsf{YES}] \leq \varepsilon$$

[1] We do not consider multiplicities although our design can be made to handle them by using techniques from counting filters [6, 10, 41, 43].

We note a detail in the above definition that has important technical consequences and that is, the false positive rate is not computed with respect to the input distribution (i.e., the probability of a false positive only depends on the internal randomness of the filter and not the random process of drawing the input set). As a consequence, we can argue about filter designs that work over all input sets except some that occur very rarely under $\mathcal{P}_n$. This is stronger than saying that $\mathcal{F}$ works only on average over $\mathcal{P}_n$. Moreover, we also want designs that do not require knowing the specific realization of the input set in advance. Our dependency on $\mathcal{P}_n$ shows up in the space requirements of the filter.

**Access to $\mathcal{P}$ and $\mathcal{Q}$.** For simplicity, we consider filter designs that have oracle access to $\mathcal{P}$ and $\mathcal{Q}$: upon seeing a key $x$, we also get $p_x$ and $q_x$. We assume that this is done in constant time and do not account for the size of the oracle when we bound the size of the filter. Critics of this model have argued that assuming oracle access to a distribution over the universe is too strong of an assumption. Indeed, this is a valid concern, since we are talking about a data structure that is meant to save space over a dictionary. We try to alleviate this concern in several ways. On one hand, our construction can tolerate mistakes. In particular, our designs are robust even if we have a constant factor approximation for $p_x$ and $q_x$, in the sense in which the space increases only by $O(n)$ bits and the time to perform each operation by an added constant. The assumption of access to such approximate oracles is standard [13, 23] and can be based on samples of historical information, on frequency estimators such as Count-Min [17] or Count-Sketch [16], or on machine learning models (see for instance, the neural-net based frequency predictor of Hsu et al. [32]). This view is indeed part of an emerging body of work on algorithms with predictions, to which the data structure perspective is just beginning to contribute [14, 18, 27–30, 36, 37, 48].

On the other hand, empirical studies have shown that significant gains are possible even when using off-the-shelf, "simplistic" learned components such as random forest classifiers. In particular, the Partitioned Learned Bloom Filter [48] and the Adaptive Learned Bloom Filter [18] consider settings in which the size of the learned component is comparable to the size of the filter itself (rather than proportional to the size of the universe), and compare the traditional Bloom filter design [9] with a learned design whose space includes the random forest classifier. In one experiment with a universe of $\approx 138,000$ keys and a classifier of 136Kb, [18] show that, within the range 150-300Kb, there is a 98% decrease in false positive rate compared to the original Bloom filter. This continues to hold for larger universe ($\approx 450,000$ keys) with total allocated space between 200Kb and 1000Kb. A discussion of how our current (theoretical) design compares to the ones in [18] and [48] can be found in Section 1.2.

Finally, strictly speaking, our designs do not necessarily rely on knowing $p_x$ and $q_x$ for every element inserted or queried. As we will see in the next section, our designs depend rather on knowing which subset of the universe a key $x$ belongs to. This corresponds to a partitioning of the universe that mainly depends on the ratio $q_x/p_x$, rather than the individual values of $p_x$ and $q_x$ (with the exception of values of $p_x$ and $q_x$ that are very small, e.g., smaller than $1/n$). This can conceivably lead to even smaller oracles that just output the partition to which an element belongs. We also do not need to query the entire universe in order to set the internal parameters of the filter, in contrast to [18, 48].

**Weighted Bloom filters.** The design that we propose starts by gathering information about the input and query distributions, using $\mathsf{polylog}(n)$ samples.[2] This information is used to estimate the internal parameters of the filter which are then used to allocate space for the filter and implement the query and insert operations. Thus, the most important aspect of our design is in setting the aforementioned internal parameters.

[2] Elements that are inserted in the set during that time can be stored in a small dictionary that only requires $\mathsf{polylog}(n)$ bits, see Section 4.3.

As a baseline for comparison, we can consider the classic Bloom filter design which allocates an array of $\approx 1.44 \cdot n \log(1/\varepsilon)$ bits and hashes every key to $\log(1/\varepsilon)$ locations in the array. Upon insertion, the corresponding bits are set to 1 and a query returns a YES if and only if all locations are set to 1. The more locations we hash into, the lower the probability that we make a mistake. Thus, a natural approach for our problem would be to vary the number of hashed locations of $x$ based on $p_x$ and $q_x$. Indeed, this is the question investigated by Bruck, Gao and Jiang [12] in their Weighted Bloom filter design. More precisely, let $k_x$ denote the number of locations that key $x$ is hashed to. Then [12] investigates the optimal choice of the parameters $k_x$ that limits the false positive rate in expectation over both the input and the query distribution. Their approach follows the original Bloom filter analysis and casts the problem as an unconstrained optimization problem in which $k_x$ is allowed to be any real number (including negative). For more details, we refer the reader to Section 1.2 This formulation and the fact that their false positive rate is taken as an average over $\mathcal{P}_n$ leads to situations in which $k_x$ can be made arbitrarily large and, with high probability, the filter is filled with 1s and has a high false positive probability (for instance, when a key is queried very rarely). To avoid such situations, as we shall see next, optimal choices for $k_x$ exhibit some rather counter-intuitive trade-offs between $p_x$ and $q_x$.

## 1.1 Our Contributions

We start by discussing near-optimal choices for $k_x$ for a Weighted Bloom filter that is a $(\mathcal{Q}, \varepsilon)$-filter for sets drawn from $\mathcal{P}_n$. While this filter is not the most efficient of the filters we construct, reasoning through it helps us present our parametrizations and addresses the fact that Bloom filters remain well-liked in practice [35]. Specifically, we define $k_x$ as follows:[3]

$$k_x \triangleq \begin{cases} 0 & \text{if or } p_x > 1/n \text{ or } q_x \le \varepsilon p_x \ , \\ \log(1/\varepsilon \cdot q_x/p_x) & \text{if } \varepsilon p_x < q_x \le \min\{p_x, \varepsilon/n\} \ , \\ \log(1/\varepsilon) & \text{if } q_x > p_x \text{ and } p_x \le \varepsilon/n \ , \\ \log(1/(np_x)) & \text{if } q_x > \varepsilon/n \text{ and } \varepsilon/n < p_x \le 1/n \ . \end{cases}$$

The first case covers the situation in which $x$ is very likely to be included in the set or is queried very rarely (relative to $p_x$). Intuitively, it makes sense in these cases to always say YES when queried. Thus, we set $k_x = 0$ and store no information about these keys. Conversely, the third case considers the case in which $x$ is queried so often (relative to $p_x$) that we need to explicitly keep the false positive probability below $\varepsilon$, which is achieved by setting $k_x = \log(1/\varepsilon)$. This is the largest number of hash functions we employ for any key, so in this sense, we are never worse than the classical Bloom filter. The second case interpolates smoothly between the first and third cases for elements that are rarely (but not very rarely) queried (compared to how likely they are to be inserted). Finally, the fourth case interpolates between the first and third case for elements that are not too rarely queried, in which case the precise query probability does not matter. See Figure 1 for a visualization of these regimes.

To further make sense of these regimes, we consider the case of uniform queries, i.e., $q_x = 1/u$, and assume that $\varepsilon > n/u$, a standard assumption in filter design (otherwise, the filter would essentially have to answer correctly on all queries and the lower bound of $n\log_2(1/\varepsilon) - O(1)$ would not hold [15,20]). Then in the two extremes, we would set $k_x = 0$ for elements with $p_x \ge 1/(u\varepsilon)$ (first case) and $k_x = \log(1/\varepsilon)$ when $p_x \le 1/u$ (third case). Keys

[3] Throughout the paper, we employ $\log x$ to denote $\log_2 x$ and $\ln x$ to denote $\log_e x$.

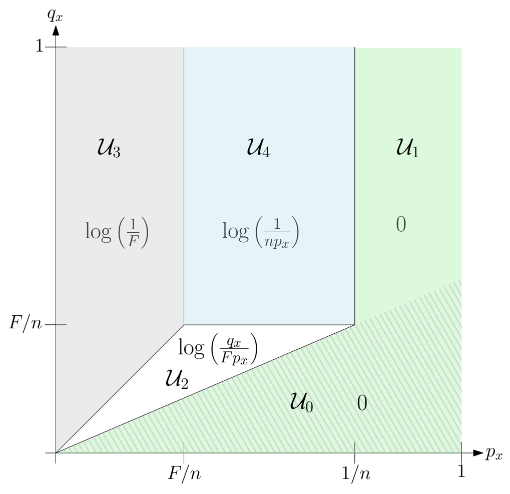

**Figure 1** A schematic visualization of the different regimes for $k_x$.

with $p_x$ between the two cases would exhibit the smooth interpolation $k_x = \log(1/(u\varepsilon)\cdot 1/p_x)$, corresponding to the intuition that the more likely an element is to be inserted, the less information we should store about it (i.e., smaller $k_x$).

**The lower bound.** Given the above parameters, we then define the quantity

$$\mathsf{LB}(\mathcal{P}_n, \mathcal{Q}, \varepsilon) \triangleq \sum_{x\in\mathcal{U}} p_x k_x$$

and show that, perhaps surprisingly, it gives a lower bound for the expected space that any $(\mathcal{Q}, \varepsilon)$-filter requires when the input set is drawn from $\mathcal{P}_n$:

▶ **Theorem 2** (Lower bound - simplified). *Let $A$ be an algorithm and assume that for any input set $S \subseteq \mathcal{U}$ with $|S| \le n$, $A(S)$ is a $(\mathcal{Q}, \varepsilon)$-filter for $S$. Then the expected size of $A(S)$ must satisfy*

$$\mathbb{E}_{\mathcal{P}_n,A}\left[|A(S)|\right] \ge \mathit{LB}(\mathcal{P}_n, \mathcal{Q}, \varepsilon) - 1 - 6n ,$$

*where $S$ is sampled with respect to $\mathcal{P}_n$ and the queries are sampled with respect to $\mathcal{Q}$.*

Previous approaches for filter lower bounds show that there exists a set $S \subseteq U$ of size $n$ for which the filter needs to use $n\log_2(1/\varepsilon) - O(1)$ bits [15, 20]. This type of lower bound is still true in our model but it does not necessarily say anything meaningful, since the bad set $S$ could be sampled in $\mathcal{P}_n$ with a negligible probability. Indeed, if we were to ignore the input distribution, then we would not be able to beat the worst input distribution and, in particular, we would need to use at least $\sup_{\mathcal{P}} \mathsf{LB}(\mathcal{P}_n, \mathcal{Q}, \varepsilon) = \mathsf{LB}(\mathcal{Q}_n, \mathcal{Q}, \varepsilon) = n\log(1/\varepsilon)$ bits in expectation, where $\mathcal{Q}_n$ denotes a distribution over $n$ independent draws from $\mathcal{Q}$.

In our model, it is therefore more natural to lower bound the *expected* size of the filter over the randomness of the input set. Finally, we remark that the full lower bound we prove is slightly stronger in that it holds for all but an unlikely collection of possible input sets, i.e. we only require that $A(S)$ is a $(\mathcal{Q}, \varepsilon)$-filter for $S \in \mathcal{T}$ where $\mathcal{T} \subseteq \mathbb{P}(U)$ and $\Pr_{\mathcal{P}_n}\left[S \notin \mathcal{T}\right] \le \frac{1}{\log u}$ (see Theorem 6).

**The space-efficient filter.** We also show a filter design that asymptotically matches our space lower bound and executes operations in constant time in the worst case:

▶ **Theorem 3** (Space-efficient filter – simplified)**.** *Given $0 < \varepsilon < 1$, there is a $(\mathcal{Q}, \varepsilon)$-filter with the following guarantees:*

- *it is a $(\mathcal{Q}, \varepsilon)$-filter with high probability over sets drawn from $\mathcal{P}_n$, if $\sum_{x \in \mathcal{U}} p_x q_x \leq \varepsilon/n$,*
- *queries and insertions take constant time in the worst case,*
- *the space it requires is $(1 + o_n(1)) \cdot \mathit{LB}(\mathcal{P}_n, \mathcal{Q}, \varepsilon) + O(n)$ bits.*

The construction uses the $k_x$ values from above in conjunction with the fingerprinting technique of Carter *et al.* [15] to obtain results that are comparable to state-of-the-art (classic) filter implementations that execute all operations (queries and insertions) in worst case constant time, and are space efficient, in the sense in which they require $(1 + o_n(1)) \cdot n \log(1/\varepsilon) + O(n)$ bits [1, 4–6]. The condition that $\sum_{x \in \mathcal{U}} p_x q_x \leq \varepsilon/n$ can be seen as a generalization of the standard filter assumption that $\varepsilon \geq n/u$.

**The Daisy Bloom filter.** For completeness, we also present our variant of the Weighted Bloom filter, which we call the *Daisy Bloom filter*:[4]

▶ **Theorem 4** (Daisy Bloom filter – simplified)**.** *Given $0 < \varepsilon < 1$, the Daisy Bloom filter has the following guarantees:*

- *it is a $(\mathcal{Q}, \varepsilon)$-filter with high probability over sets drawn from $\mathcal{P}_n$, if $\sum_{x \in \mathcal{U}} p_x q_x \leq \varepsilon/n$,*
- *queries and insertions take at most $\lceil \log_2(1/\varepsilon) \rceil$ time in the worst case,*
- *the space it requires is $\log(e) \cdot \mathit{LB}(\mathcal{P}_n, \mathcal{Q}, \varepsilon) + O(n)$ bits.*

In contrast to the weighted Bloom filters of Bruck et al. [12], the Daisy Bloom filter executes operations in time that is at most $\lceil \log_2(1/\varepsilon) \rceil$ in the worst case (versus arbitrarily large) and achieves a false positive rate of at most $\varepsilon$ with high probability over the input set (and not just on average). We also depart in our analysis from their unconstrained optimization approach (to setting $k_x$ ) and instead use Bernstein's inequality to argue that, if the length of the array is set to $\log(e) \cdot \mathsf{LB}(\mathcal{P}_n, \mathcal{Q}, \varepsilon) + O(n)$ bits, then whp, at most half of the entries in the array will be set to 1 (see Section 5 for more details).

## 1.2 Related Work

Filters have been studied extensively in the literature [2, 5, 6, 15, 20, 34, 41, 42, 45], with Bloom filters perhaps the most widely employed variants in practice [35]. Learning-based approaches to classic algorithm design have recently attracted a great deal of attention, see e.g. [21, 31–33, 46]. For a comprehensive survey on learned data structures, we refer the reader to Ferragina and Vinciguerra [29].

### Weighted Bloom Filters

Given information about the probability of inserting and querying each element, Bruck, Gao and Jiang [12] set out to find an optimal choice of the parameters $k_x$ that limit the false positive rate (in expectation over both the input and the query distribution). The approach

[4] The daisy is one of our favorite flowers, especially when in full bloom, and is also a subsequence of "*dy*n*a*m*i*c *s*trech*y*" which describes the key properties of our data structure. It is also the nickname of the Danish queen, whose residence is not far from the place where this work was conceived. Daisy Bloom filters are not related to any celebrities.

is to solve an unconstrained optimization problem where the variables $k_x$ can be any real number. In a post-processing step each $k_x$ is rounded to the nearest non-negative integer. Unfortunately, this process does not lead to an optimal choice of parameters, and in fact, does not guarantee a non-trivial false positive rate. The issue is that the solution to the unconstrained problem may have many negative values of $k_x$, so even though the weighted sum $\sum_x p_x k_x$ is bounded, the post-processed sum $\sum_x p_x \max(k_x, 0)$ can be arbitrarily large. In particular, this is the case if at least one element is queried very rarely. This means that the weighted Bloom filter may consist only of 1s with high probability, resulting in a false positive probability of 1.

The above issue was noted by Wang, Ji, Dang, Zheng and Zhao [49] who attempt to correct the values for $k_x$, but their analysis still suffers from the same, more fundamental, problem: the existence of a very rare query element drives the false positive rate to 1. Wang et al. [49] also show an information-theoretical “approximate lower bound” on the number of bits needed for a weighted Bloom filter with given distributions $\mathcal{P}$ and $\mathcal{Q}$. The sense in which the lower bound is approximate is not made precise, and the lower bound is certainly not tight (for example, it can be negative).

### Partitioned Learned Bloom Filters

There are several learned Bloom filter designs that assume that the filter has access to a learned model of the input set [18,33,37,48]. The model is given a fixed input set $S$ and a representative sample of elements in $\mathcal{U} \setminus S$ ( the query distribution is not specified). Given a query element $x$, the model returns a *score* $s(x) \in [0, 1]$, which can be intuitively thought of as the model's belief that $x \in S$. Based on this score, Vaidya, Knorr, Mitzenmacher and Kraska [48] choose a fixed number of $k$ thresholds, partition the elements according to these thresholds, and build separate Bloom filters for each set of the partition. For fixed threshold values, they then formulate the optimization problem of setting the false positive rates $f_i$ such that the total space of the data structure is minimized and the overall false positive rate is at most a given $F$.

As noted by Ferragina and Vinciguerra [29], a significant drawback in these constructions is that the guarantees they provide depend significantly on the query set given as input to the machine learning component and in particular, the set being representative for the whole query distribution. We avoid this issue by making the dependencies on $q_x$ explicit and by bounding the average false positive probability even when just one element is queried. In addition, our data structure does not need to know the set $S$ in advance (and hence, training can be done just once, in a pre-processing phase), employs only one data structure, and our guarantees are robust to approximate values for $p_x$ and $q_x$.

## 1.3 Paper Organization

After some preliminaries, Section 3 shows our lower bound on the space usage. In Section 4, we discuss a space-efficient filter with constant time worst-case operations. Finally, Section 5 presents the analysis of the Daisy Bloom filter.

## 2 Preliminaries

For clarity, throughout the paper, we will distinguish between probabilities over the randomness of the input set, denoted by $\Pr_{\mathcal{P}_n}[\cdot]$, and probabilities over the internal randomness of the filter, denoted by $\Pr_A[\cdot]$. Joint probabilities are denoted by $\Pr_{\mathcal{P}_n,A}[\cdot]$. For the analysis, it will also make sense to partition the universe $\mathcal{U}$ into the following 5 parts:

$$\begin{aligned}
\mathcal{U}_0 &\triangleq \{x \in \mathcal{U} \mid q_x \le \varepsilon p_x\} \ , \\
\mathcal{U}_1 &\triangleq \{x \in \mathcal{U} \mid q_x > \varepsilon p_x \text{ and } p_x > 1/n\} \ , \\
\mathcal{U}_2 &\triangleq \{x \in \mathcal{U} \mid \varepsilon p_x < q_x \le \min\{p_x, \varepsilon/n\}\} \ , \\
\mathcal{U}_3 &\triangleq \{x \in \mathcal{U} \mid q_x > p_x \text{ and } \varepsilon/n \ge p_x\} \ , \\
\mathcal{U}_4 &\triangleq \{x \in \mathcal{U} \mid q_x > \varepsilon/n \text{ and } \varepsilon/n < p_x \le 1/n\} \ .
\end{aligned}$$

The high probability guarantees we obtain increase with $\mathsf{LB}(\mathcal{P}_n, \mathcal{Q}, \varepsilon)$. Therefore, such bounds are meaningful for distributions in which the optimal size $\mathsf{LB}(\mathcal{P}_n, \mathcal{Q}, \varepsilon)$ of a filter is not too small. Similarly, we can assume that the size of the universe is polynomial in $n$, and so $\log(1/\varepsilon) = O(\log n)$ in the standard case in which $\varepsilon > n/|\mathcal{U}|$. Therefore, while in general $\mathsf{LB}(\mathcal{P}_n, \mathcal{Q}, \varepsilon)$ can be much smaller than $n \log_2(1/\varepsilon)$, we do require some mild dependency on $n$ for the high probability bounds to be meaningful. Finally, we recall the following classic result in data compression:

▶ **Theorem 5** (Kraft's inequality [47]). *For any instantaneous code (prefix code) over an alphabet of size $D$, the codeword lengths $\ell_1, \ell_2, \ldots, \ell_m$ must satisfy the inequality*

$$\sum_i D^{-\ell_i} \le 1 \ .$$

*Conversely, given a set of codeword lengths that satisfy this inequality, there exists an instantaneous code with these word lengths.*

## 3 The Lower Bound

The goal of this section is to prove the lower bound from Theorem 2. As discussed, we prove a slightly stronger statement where we allow our algorithm to not produce a $(\mathcal{Q}, \varepsilon)$-filter for some input sets as long as the probability of sampling them is low. Formally, we show that:

▶ **Theorem 6.** *Let $\mathcal{T} \subseteq \mathbb{P}(\mathcal{U})$ be given such that $\Pr_{\mathcal{P}_n}[S \notin \mathcal{T}] \le \frac{1}{\log u}$. If $A$ is an algorithm such that for all $S \in \mathcal{T}$, $A(S)$ is a $(\mathcal{Q}, \varepsilon)$-filter for $S$. Then the expected size of $A(S)$ must satisfy*

$$\mathbb{E}_{\mathcal{P}_n, A}[|A(S)|] \ge \mathit{LB}(\mathcal{P}_n, \mathcal{Q}, \varepsilon) - 1 - 6n \ ,$$

*where $S$ is sampled with respect to $\mathcal{P}_n$.*

**Proof.** Each instance $\mathcal{I}$ of the data structure corresponds to a subset $\mathcal{U}_\mathcal{I} \subset \mathcal{U}$ on which the data structure answers YES. We denote the number of bits needed by such an instance by $|\mathcal{I}|$. For any set $S \in \mathcal{T}$, we have that $\mathcal{I} = A(S)$ satisfies that $S \subseteq \mathcal{U}_\mathcal{I}$ and

$$\mathbb{E}_A\left[\sum_{x \in \mathcal{U}_\mathcal{I} \setminus S} q_x\right] \le \varepsilon \ .$$

The goal is to prove that

$$\mathbb{E}_{\mathcal{P}_n, A}[|A(S)|] \ge n \cdot \left(\sum_{x \in \mathcal{U}_2} p_x \log\left(\frac{1}{\varepsilon} \cdot \frac{q_x}{p_x}\right) + \sum_{x \in \mathcal{U}_3} p_x \log \frac{1}{\varepsilon} + \sum_{x \in \mathcal{U}_4} p_x \log \frac{1}{n p_x}\right) - 1 - 6n \ .$$

We will lower bound $\mathbb{E}_{\mathcal{P}_n,A}[|A(S)|]$ by using it to encode an ordered sequence of $n$ elements drawn according to $\mathcal{P}_n$. Specifically, for any ordered sequence of $n$ elements $\hat{S} \in \mathcal{U}^n$, we let $S \subseteq U$ be the set of distinct elements and let $\mathcal{I} = A(S)$ as above. We first note that to encode $\hat{S} \sim \mathcal{P}_n$, in expectation, we need at least the entropy number of bits, i.e.,

$$n \sum_{x \in \mathcal{U}} p_x \log \frac{1}{p_x} \ . \tag{1}$$

Now our encoding using $\mathcal{I}$ will depend on whether $S \in \mathcal{T}$ or not. First, we will use 1 bit to describe whether $S \in \mathcal{T}$ or not. For $(x_i)_{i\in[n]} \in \hat{S}$, we will denote $b_i$ to be the number bits to encode $x_i$. If $S \notin \mathcal{T}$ then for all $i \in [n]$ we encode $x_i$ using $b_i = \lceil \log(1/p_{x_i}) \rceil$ bits. If $S \in \mathcal{T}$ then for all $i \in [n]$ we encode $x_i$ depending on which subset if $\mathcal{U}$ it belongs to:

1. If $x_i \in \mathcal{U}_0 \cup \mathcal{U}_1$, we encode $x_i$ using $b_i = \lceil \log(4/p_{x_i}) \rceil$ bits.
2. If $x_i \in \mathcal{U}_2$, we encode $x_i$ using $b_i = \left\lceil \log\left(4 \frac{\sum_{y\in\mathcal{U}_\mathcal{I}\cap\mathcal{U}_2} q_y}{q_{x_i}}\right)\right\rceil$ bits.
3. If $x_i \in \mathcal{U}_3$, we encode $x_i$ using $b_i = \left\lceil \log\left(4 \frac{\sum_{y\in\mathcal{U}_\mathcal{I}\cap\mathcal{U}_3} p_y}{p_{x_i}}\right)\right\rceil$ bits.
4. If $x_i \in \mathcal{U}_4$, we encode $x_i$ using $b_i = \lceil \log(4\,|\mathcal{U}_\mathcal{I} \cap \mathcal{U}_4|) \rceil$ bits.

It is clear from the construction that we satisfy the requirement for Theorem 5 thus there exists such an encoding. Now we will bound the expectation of the size of this encoding:

$$\mathbb{E}_{\mathcal{P}_n,A}\left[|A(S)| + 1 + \sum_{i\in[n]} b_i\right] = \mathbb{E}_{\mathcal{P}_n,A}[|A(S)|] + 1 + \mathbb{E}_{\mathcal{P}_n,A}\left[\sum_{i\in[n]} b_i\right] \ .$$

We will write $\mathbb{E}_{\mathcal{P}_n,A}\left[\sum_{i\in[n]} b_i\right] = \mathbb{E}_{\mathcal{P}_n,A}\left[[S\in\mathcal{T}]\sum_{i\in[n]} b_i\right] + \mathbb{E}_{\mathcal{P}_n,A}\left[[S\notin\mathcal{T}]\sum_{i\in[n]} b_i\right]$, and bound each term separately.

We start by bounding $\mathbb{E}_{\mathcal{P}_n,A}\left[[S\notin\mathcal{T}]\sum_{i\in[n]} b_i\right]$.

$$\begin{aligned}
\mathbb{E}_{\mathcal{P}_n,A}\left[[S\notin\mathcal{T}]\sum_{i\in[n]} b_i\right] &= \mathbb{E}_{\mathcal{P}_n}\left[[S\notin\mathcal{T}]\sum_{i\in[n]} \lceil \log(1/p_{x_i}) \rceil\right] \\
&\le \Pr_{\mathcal{P}_n}[S\notin\mathcal{T}]\, n + \mathbb{E}_{\mathcal{P}_n}\left[[S\notin\mathcal{T}] \log\left(\prod_{i\in[n]} 1/p_{x_i}\right)\right] \\
&= \Pr_{\mathcal{P}_n}[S\notin\mathcal{T}]\, n + \sum_{\hat{s}\in\mathcal{U}^n} [\hat{s}\in\mathcal{T}] \Pr_{\mathcal{P}_n}\left[\hat{S}=\hat{s}\right] \log \frac{1}{\Pr_{\mathcal{P}_n}\left[\hat{S}=\hat{s}\right]}
\end{aligned}$$

Now using Jensen's inequality we get that

$$\sum_{\hat{s}\in\mathcal{U}^n} [\hat{s}\in\mathcal{T}] \Pr_{\mathcal{P}_n}\left[\hat{S}=\hat{s}\right] \log \frac{1}{\Pr_{\mathcal{P}_n}\left[\hat{S}=\hat{s}\right]} \le \Pr_{\mathcal{P}_n}[S\notin T] \log\left(\tfrac{1}{\Pr_{\mathcal{P}_n}[S\notin T] u^n}\right) \ .$$

Putting this together with the fact that $\Pr_{\mathcal{P}_n}[S\notin T] \le \frac{1}{\log u}$, we get that,

$$\mathbb{E}_{\mathcal{P}_n,A}\left[[S\notin\mathcal{T}]\sum_{i\in[n]} b_i\right] \le 2n \ .$$

Now we bound $\mathbb{E}_{\mathcal{P}_n,A}\left[[S \in \mathcal{T}]\sum_{i\in[n]} b_i\right] = \sum_{i\in[n]} \mathbb{E}_{\mathcal{P}_n,A}\left[[S \in \mathcal{T}]\, b_i\right]$. We will bound $\mathbb{E}_{\mathcal{P}_n,A}\left[[S \in \mathcal{T}]\, b_i\right]$ depending on which subset of $\mathcal{U}$ that $x_i$ belongs to.

If $x_i \in \mathcal{U}_0 \cup \mathcal{U}_1$, then we have that $\mathbb{E}_{\mathcal{P}_n,A}\left[[S \in \mathcal{T}]\, b_i\right] \le \lceil \log(4/p_{x_i})\rceil \le 3 + \log(1/p_{x_i})$.

If $x_i \in \mathcal{U}_2$, define $Z_2 = \mathcal{U}_\mathcal{I} \cap \mathcal{U}_2$. Then

$$\mathbb{E}_{\mathcal{P}_n,A}\left[[S \in \mathcal{T}]\, b_i\right] \le 3 + \mathbb{E}_{\mathcal{P}_n,A}\left[[S \in \mathcal{T}]\log\left(\frac{\sum_{y\in Z_2} q_y}{q_{x_i}}\right)\right] .$$

We know that $\sum_{y\in S\cap\mathcal{U}_2} q_y \le \varepsilon$ since $q_y \le \varepsilon/n$ for all $y \in \mathcal{U}_2$ and $|S| \le n$. We also know that $\mathbb{E}_A\left[\sum_{x\in Z_2\setminus S} q_x\right] \le \varepsilon$ for $S \in \mathcal{T}$. Now using Jensen's inequality we get that

$$\begin{aligned}\mathbb{E}_{\mathcal{P}_n,A}\left[[S \in \mathcal{T}]\log\left(\frac{\sum_{y\in Z_2} q_y}{q_{x_i}}\right)\right] &= \mathbb{E}_{\mathcal{P}_n}\left[[S \in \mathcal{T}]\,\mathbb{E}_A\left[\log\left(\frac{\sum_{y\in Z_2} q_y}{q_{x_i}}\right)\right]\right] \\ &\le \mathbb{E}_{\mathcal{P}_n}\left[[S \in \mathcal{T}]\log\left(\frac{\mathbb{E}_A\left[\sum_{y\in Z_2} q_y\right]}{q_{x_i}}\right)\right] \\ &\le \mathbb{E}_{\mathcal{P}_n}\left[[S \in \mathcal{T}]\log\left(\frac{2\varepsilon}{q_{x_i}}\right)\right] \le 1 + \mathbb{E}_{\mathcal{P}_n}\left[\log\left(\frac{\varepsilon}{q_{x_i}}\right)\right] .\end{aligned}$$

If $x_i \in \mathcal{U}_3$, define $Z_3 = \mathcal{U}_\mathcal{I} \cap \mathcal{U}_3$. Then

$$\mathbb{E}_{\mathcal{P}_n,A}\left[[S \in \mathcal{T}]\, b_i\right] \le 3 + \mathbb{E}_{\mathcal{P}_n,A}\left[[S \in \mathcal{T}]\log\left(\frac{\sum_{y\in Z_3} p_y}{p_{x_i}}\right)\right] .$$

We know that $\sum_{y\in S\cap\mathcal{U}_3} p_y \le \varepsilon$ since $p_y \le \varepsilon/n$ for all $y \in \mathcal{U}_3$ and $|S| \le n$. We also know that $\mathbb{E}_A\left[\sum_{x\in Z_3\setminus S} p_x\right] \le \mathbb{E}_A\left[\sum_{x\in Z_3\setminus S} q_x\right] \le \varepsilon$ for $S \in \mathcal{T}$. Using Jensen's inequality we get that,

$$\begin{aligned}\mathbb{E}_{\mathcal{P}_n,A}\left[[S \in \mathcal{T}]\log\left(\frac{\sum_{y\in Z_3} p_y}{p_{x_i}}\right)\right] &= \mathbb{E}_{\mathcal{P}_n}\left[[S \in \mathcal{T}]\,\mathbb{E}_A\left[\log\left(\frac{\sum_{y\in Z_3} p_y}{p_{x_i}}\right)\right]\right] \\ &\le \mathbb{E}_{\mathcal{P}_n}\left[[S \in \mathcal{T}]\log\left(\frac{\mathbb{E}_A\left[\sum_{y\in Z_3} p_y\right]}{p_{x_i}}\right)\right] \\ &\le \mathbb{E}_{\mathcal{P}_n}\left[[S \in \mathcal{T}]\log\left(\frac{2\varepsilon}{p_{x_i}}\right)\right] \le 1 + \mathbb{E}_{\mathcal{P}_n}\left[\log\left(\frac{\varepsilon}{p_{x_i}}\right)\right] .\end{aligned}$$

If $x_i \in \mathcal{U}_4$, define $Z_4 = \mathcal{U}_\mathcal{I} \cap \mathcal{U}_4$. Then $\mathbb{E}_{\mathcal{P}_n,A}\left[[S \in \mathcal{T}]\, b_i\right] \le 3 + \mathbb{E}_{\mathcal{P}_n,A}\left[[S \in \mathcal{T}]\log\left(|Z_4|\right)\right]$. We know that $|Z_4| = |Z_4 \cap S| + |Z_4 \setminus S| \le n + \frac{n}{\varepsilon}\sum_{x\in Z_4\setminus S} q_x$ since $q_y > \varepsilon/n$ for all $y \in \mathcal{U}_4$ and $|S| \le n$. Using Jensen's inequality we get that, for $Z_4' = Z_4 \setminus S$:

$$\begin{aligned}\mathbb{E}_{\mathcal{P}_n,A}\left[[S \in \mathcal{T}]\log\left(n + \frac{n}{\varepsilon}\sum_{x\in Z_4'} q_x\right)\right] &= \mathbb{E}_{\mathcal{P}_n}\left[[S \in \mathcal{T}]\,\mathbb{E}_A\left[\log\left(n + \frac{n}{\varepsilon}\sum_{x\in Z_4'} q_x\right)\right]\right] \\ &\le \mathbb{E}_{\mathcal{P}_n}\left[[S \in \mathcal{T}]\log\left(\mathbb{E}_A\left[n + \frac{n}{\varepsilon}\sum_{x\in Z_4'} q_x\right]\right)\right] \\ &\le \mathbb{E}_{\mathcal{P}_n}\left[[S \in \mathcal{T}]\log(2n)\right] \le 1 + \log(n) .\end{aligned}$$

Combining it all we get an encoding that in expectation uses at most

$$\mathbb{E}_{\mathcal{P}_n,A}\left[|A(S)|\right] + 1 + 6n + \sum_{x\in(\mathcal{U}_0\cup\mathcal{U}_1)} p_x\log(1/p_x) + \sum_{x\in\mathcal{U}_2} p_x\log(\varepsilon/q_x) + \sum_{x\in\mathcal{U}_3} p_x\log(\varepsilon/p_x) + \sum_{x\in\mathcal{U}_4} p_x\log n .$$

bits to encode $\hat{S}$. Comparing this with Equation (1) we get that,

$$\mathbb{E}_{\mathcal{P}_n,A}\left[|A(S)|\right] \geq n \cdot \left( \sum_{x\in\mathcal{U}_2} p_x \log\left(\frac{1}{\varepsilon}\cdot\frac{q_x}{p_x}\right) + \sum_{x\in\mathcal{U}_3} p_x \log\frac{1}{\varepsilon} + \sum_{x\in\mathcal{U}_4} p_x \frac{1}{np_x} \right) - 1 - 6n \ .$$

This proves the claim. ◀

## 4 Space-Efficient Filter

In this section, we show how one can use the $k_x$ values proposed to design a space-efficient $\mathcal{Q}$-filter with worst-case constant time operations. Formally, we show that:

▶ **Theorem 7.** *Assume that $\mathcal{P}_n$ and $\mathcal{Q}$ satisfy $n\sum_{x\in\mathcal{U}} p_x q_x \leq \varepsilon$. Then there exists a filter with the following guarantees:*

- *there exists $\mathcal{T} \subseteq \mathbb{P}(U)$ where a set $S \in \mathcal{T}$ with high probability over the randomness of $\mathcal{P}_n$, such that the filter is a $(\mathcal{Q},\varepsilon)$-filter for any $S \in \mathcal{T}$,*
- *the filter uses $(1 + o_n(1)) \cdot LB(\mathcal{P}_n, \mathcal{Q}, \varepsilon) + O(n)$ bits,*
- *the filter executes queries and insertions in worst case constant time and,*
- *the filter does not fail with high probability over its internal randomness.*

### 4.1 Construction

For $j \in \{1, \ldots, \lceil\log(1/\varepsilon)\rceil\}$, we let $\mathcal{U}^{(j)} \triangleq \{x \in \mathcal{U} \mid \lceil k_x\rceil = j\}$ denote the set of elements that hash to $j$ locations in the Daisy Bloom filter and $P_j \triangleq \sum_{x\in\mathcal{U}^{(j)}} p_x$ denote the probability that we select an element from $\mathcal{U}^{(j)}$ in one sample from $\mathcal{P}$. Then $n_j \triangleq n \cdot P_j$ denotes the average number of elements from $\mathcal{U}^{(j)}$ that we expect to see in the input set. We distinguish between the sets $\{\mathcal{U}^{(j)}\}$ depending on their corresponding $n_j$. Specifically, we say that $\mathcal{U}^{(j)}$ is a *rare class* if $n_j < n/\log^c n$, for some constant $c > 2$, and otherwise we say that $\mathcal{U}^{(j)}$ is a *frequent class*. We further define $\mathcal{U}_r \subseteq \mathcal{U}$ to be the set of all elements that are in a rare class, i.e., $\mathcal{U}^{(j)} \subseteq \mathcal{U}$ if and only if $\mathcal{U}^{(j)}$ is a rare class.

Now let $\mathcal{F}(\varepsilon, n)$ be a (standard) filter implementation for at most $n$ elements with false positive probability at most $\varepsilon$. We focus on implementations that execute all operations (queries and insertions) in worst-case constant time, and are space efficient: they require $(1 + o_n(1)) \cdot n\log(1/\varepsilon) + O(n)$ bits [1,4–6]. We employ $\lceil\log(1/\varepsilon)\rceil + 1$ instantiations of $\mathcal{F}$, which we denote by $\mathcal{F}_1, \ldots, \mathcal{F}_{\lceil\log(1/\varepsilon)\rceil}$ and $\mathcal{F}_r$. They are parametrized as follows: for $j \in \{1, \ldots, \lceil\log(1/\varepsilon)\rceil\}$, we further define $N_j \triangleq (1+1/\log n)\cdot n_j$ and instantiate $\mathcal{F}_j = \mathcal{F}(2^{-j}, N_j)$. We instantiate $\mathcal{F}_r$ as $\mathcal{F}_r = \mathcal{F}(F, N_r)$, where $N_r \triangleq \Theta(n/\log^{c-1} n)$.

**Operations.** We distinguish between elements that are in a frequent class and elements that are in a rare class. If an element is in a frequent class $\mathcal{U}^{(j)}$, then operations are forwarded to the corresponding filter $\mathcal{F}_j$. Otherwise, the operation is forwarded to $\mathcal{F}_r$. Since all the filters we employ perform operations in constant time in the worst case, the same holds for our construction[5].

[5] We note here that it is possible to combine the filters $\mathcal{F}_1, \ldots, \mathcal{F}_{\lceil\log(1/\varepsilon)\rceil}$ into one single data structure. One could use, for example, the balls-into-bins implementation in [5], where elements are randomly assigned to one of $n/\Theta(\log n/(\log(1/\varepsilon)))$ buckets and the buckets explicitly store random strings of length $\log(1/\varepsilon)$ associated with the elements that hash into them. Combining $\mathcal{F}_1, \ldots, \mathcal{F}_{\lceil\log(1/\varepsilon)\rceil}$ would then entail “superimposing” their buckets.

**Space.** We now bound the total number of bits that $\mathcal{F}_1, \ldots, \mathcal{F}_{\lceil \log(1/\varepsilon) \rceil}$ and $\mathcal{F}_r$ require:

▶ **Lemma 8.** *The above filter requires $(1 + o_n(1)) \cdot \mathit{LB}(\mathcal{P}_n, \mathcal{Q}, \varepsilon) + O(n)$ bits.*

**Proof.** Recall that $\mathcal{F}_1, \ldots, \mathcal{F}_{\lceil \log(1/\varepsilon) \rceil}$ and $\mathcal{F}_r$ are instantiations of a filter $\mathcal{F}$ which requires $(1+f(n)) \cdot n \log(1/\varepsilon) + O(n)$ bits for a set of $n$ elements and false positive probability $\varepsilon$, where the function $f(n)$ satisfies $f(n) = o_n(1)$ [1,4–6]. For simplicity, we choose the implementation in [5], where $f(n) = \Theta(\log\log n/\sqrt{\log n})$. Consequently, for $j \in \{1, \ldots, \lceil \log(1/\varepsilon) \rceil\}$, the space of $\mathcal{F}_j$ is:

$$(1 + f(N_j)) \cdot N_j \log(1/2^{-j}) + O(N_j) = (1 + f(N_j)) \cdot N_j \cdot j + O(N_j)$$

bits. Since $\mathcal{F}_j$ is instantiated only for frequent classes, it follows that $n \geq N_j \geq n/\log^c n$ and hence, $f(N_j) = \Theta(f(n))$ for all $j$ with $\mathcal{U}^{(j)}$ a frequent class. Furthermore, by definition, we have that $j \leq k_x + 1$ for all $x \in \mathcal{U}^{(j)}$ and $N_j = (1+1/\log n) \cdot n_j = (1+1/\log n) \cdot n \sum_{x \in \mathcal{U}^{(j)}} p_x$. Therefore, the space that $\mathcal{F}_j$ requires can be upper bounded by

$$(1 + \Theta(f(n))) \cdot n \sum_{x \in \mathcal{U}^{(j)}} p_x k_x + O(N_j) \ .$$

Since $\sum_j N_j = (1 + 1/\log n) \cdot n$ we get that, in the worst case in which all the classes are frequent, the filters $\mathcal{F}_1, \ldots, \mathcal{F}_{\lceil \log(1/\varepsilon) \rceil}$ require

$$(1 + o_n(1)) \cdot n \sum_{x \in \mathcal{U}} p_x k_x + O(n) = (1 + o_n(1)) \cdot \mathsf{LB}(\mathcal{P}_n, \mathcal{Q}, \varepsilon) + O(n)$$

bits. The space of the final filter $\mathcal{F}_r$ is upper bounded by $\Theta(n/\log^{c-1} n) \cdot \log(1/\varepsilon) = \Theta(n/\log^{c-2} n) = o(n)$ bits for any constant $c > 2$. The claim follows. ◀

## 4.2 Analysis

In this section, we show that the filter described above does not fail whp and that it achieves a false positive probability of at most $3\varepsilon$ with respect to $\mathcal{Q}$. In our construction, there are two sources of failure: when the number of elements which we insert into each filter exceeds the maximum capacity of the filter, and when the filters themselves fail as a consequence of their internal randomness. In the latter case, we note that the failure probability of $\mathcal{F}(\varepsilon, n)$ is guaranteed to be at most $1/\mathsf{poly}(n)$, where the degree of the polynomial is a constant of our choosing [1,4–6]. Since all of the instantiations we employ have maximum capacities which are $\Omega(n/\mathsf{polylog}(n))$, we conclude that each of these separate instantiations also fails with probability at most $1/\mathsf{poly}(n)$, and therefore, by a union bound over the $\lceil \log(1/\varepsilon) \rceil + 1 = O(\log n)$ instantiations, we get that some filter fails with probability at most $1/\mathsf{poly}(n)$. We now show that the maximum capacities we set for each filter suffice.

▶ **Lemma 9.** *Whp, at most $N_r = \Theta(n/\log^{c-1} n)$ elements from $\mathcal{U}_r$ are sampled in the input set.*

**Proof.** Let $\mathcal{U}^{(j)}$ be a rare class and let $X_j$ denote the number of elements from $\mathcal{U}^{(j)}$ that we sample in the input set. By definition, the expected number of elements we see from $\mathcal{U}^{(j)}$ satisfies $\mathbb{E}_{\mathcal{P}}[X_j] = n_j < n/\log^c n$, for some constant $c > 2$. By Chernoff bound, we therefore get that:

$$\Pr[X_j > 6n/\log^c n] \leq 2^{-6n/\log^c n} \ .$$

There are at most $\lceil \log(1/\varepsilon) \rceil = O(\log n)$ possible rare classes, and so, by the union bound, the number of elements from $\mathcal{U}_r$ that we sample in the input set is at most $N_r = \Theta(n/\log^{c-1} n)$ whp. ◀

We now focus on sampling elements from a frequent class:

▶ **Lemma 10.** *Let $\mathcal{U}^{(j)}$ be a frequent class. Then, whp, at most $N_j$ elements from $\mathcal{U}^{(j)}$ are sampled in the input set.*

**Proof.** Let $X_j$ denote the number of elements from $\mathcal{U}^{(j)}$ that we sample in the input set and note that $\mathbb{E}_{\mathcal{P}}[X_j] = n_j \geq n/\log^c n$. By Chernoff:

$$\Pr[X_j > N_j] = \Pr[X_j > (1 + 1/\log n) \cdot n_j] \leq \exp(-\Theta(n/\log^{c-2} n)) \leq 1/\operatorname{poly} n\ .$$

This concludes our proof. ◀

Finally, we bound the false positive rate of the filter:

▶ **Lemma 11.** *Assume that $\mathcal{P}_n$ and $\mathcal{Q}$ satisfy $n\sum_{x\in\mathcal{U}} p_x q_x \leq \varepsilon$ and that the input set $S$ does not make $\mathcal{F}_1, \ldots, \mathcal{F}_{\lceil\log(1/\varepsilon)\rceil}$ and $\mathcal{F}_r$ fail. Then the filter described is a $(\mathcal{Q}, 3\varepsilon)$-filter on $S$.*

**Proof.** Fix an input set $S$ and denote by $A'(S,x)$ the output of the filter when queried for an element $x$. We are interested in bounding $\Pr[A'(S,x) = \mathsf{YES}]$ for an element $x \notin S$. If $x \in \mathcal{U}_r$, then we forward the query operation to $\mathcal{F}_r$, which guarantees that $\Pr[A'(S,x) = \mathsf{YES}] \leq \varepsilon$. Therefore:

$$\sum_{x\in\mathcal{U}_r} q_x \cdot \Pr[A'(S,x) = \mathsf{YES}] \leq \sum_{x\in\mathcal{U}_r} q_x \cdot \varepsilon\ ,$$

Otherwise, if $x \notin \mathcal{U}_r$, the query is forwarded to $\mathcal{F}_j$, where $j = \lceil k_x \rceil$. In this case, $\Pr[A'(S,x) = \mathsf{YES}] \leq 2^{-j} \leq 2^{-k_x}$ and we get that

$$\sum_{x\in\mathcal{U}_0\cup\mathcal{U}_2\setminus\mathcal{U}_r} q_x \cdot \Pr[A'(S,x) = \mathsf{YES}] \leq \sum_{x\in\mathcal{U}_0\setminus\mathcal{U}_r} q_x + \sum_{x\in\mathcal{U}_2\setminus\mathcal{U}_r} p_x\varepsilon \leq \sum_{x\in\mathcal{U}_0\cup\mathcal{U}_2\setminus\mathcal{U}_r} p_x \cdot \varepsilon\ ,$$

and similarly,

$$\sum_{x\in\mathcal{U}_1\cup\mathcal{U}_4\setminus\mathcal{U}_r} q_x \cdot \Pr[A'(S,x) = \mathsf{YES}] \leq \sum_{x\in\mathcal{U}_1\setminus\mathcal{U}_r} q_x + \sum_{x\in\mathcal{U}_4\setminus\mathcal{U}_r} np_xq_x \leq \sum_{x\in\mathcal{U}_1\cup\mathcal{U}_4\setminus\mathcal{U}_r} np_xq_x\ .$$

Finally, we have that

$$\sum_{x\in\mathcal{U}_3\setminus\mathcal{U}_r} q_x \cdot \Pr[A'(S,x) = \mathsf{YES}] \leq \sum_{x\in\mathcal{U}_3\setminus\mathcal{U}_r} q_x \cdot \varepsilon\ .$$

Adding all of these quantities, we obtain the claim. ◀

## 4.3 Remarks

The filter construction assumes that we know, in advance, whether a class $\mathcal{U}^{(j)}$ is frequent and, if so, what is the value of its corresponding $P_j = \sum_{x\in\mathcal{U}^{(j)}} p_x$. This is because we employ fixed capacity filters which require us to provide an upper bound on the cardinality of the input set $S \cap \mathcal{U}^{(j)}$ at all points in time. We note that this assumption can be alleviated in two ways: on one hand, one can employ filters that do not require us to know the size of the input set in advance [8, 42]. This would incur an additional $\Theta(n \log\log n)$ bits in the space consumption of our filter (operations would remain constant time worst case).

On the other hand, one can estimate $P_j$ for all frequent classes $\mathcal{U}^{(j)}$ if we are allowed to take $\operatorname{polylog}(n)$ samples from $\mathcal{P}$ before constructing the filter. Specifically, fix $j \in \{1, \ldots, \lceil\log(1/\varepsilon)\rceil\}$ and take $\ell = O(\log^{2c} n)$ samples from $\mathcal{P}$. Define $Z_j$ to be the number of sampled elements that are in $\mathcal{U}^{(j)}$. Then, if $\mathcal{U}^{(j)}$ is indeed frequent, by the standard Chernoff bound we get that, whp, $Z_j > \log^c n$ elements and $Z_j/\ell$ is an unbiased estimator for $P_j$ with the guarantee that $P_j = (1 \pm O(1/\log^{(c-1)/2} n)) \cdot Z_j/\ell$ whp. Note that we can tolerate such an estimate since we set the maximum capacity of each filter to be $N_j = (1+1/\log n)\cdot nP_j$. A similar argument can be used for estimating the lower bound $\mathsf{LB}(\mathcal{P}_n, \mathcal{Q}, \varepsilon) = n \cdot \sum_{x\in\mathcal{U}} p_x \cdot k_x$ whp. Specifically, by the definition of $P_j$, we have that

$$\sum_{x\in\mathcal{U}} p_x \cdot \lceil k_x \rceil = \sum_{j=1}^{\lceil \log(1/\varepsilon)\rceil} j \cdot P_j \ .$$

If $\mathcal{U}^{(j)}$ is a frequent class, then by the above argument we have an estimate of its $P_j$. Otherwise, we know that $P_j < 1/\log^c n$ and, since $j \leq \lceil \log(1/\varepsilon)\rceil = O(\log n)$, get that

$$\sum_{j \text{ s.t. } \mathcal{U}^{(j)}\in\mathcal{U}_r} j \cdot P_j \leq \sum_{j \text{ s.t. } \mathcal{U}^{(j)}\in\mathcal{U}_r} \lceil \log(1/\varepsilon)\rceil \cdot 1/\log^c n \leq 1/\log^{c-2} n \ ,$$

which contributes a $o(n)$ term to the lower bound.

## 5 The Daisy Bloom Filter Analysis

In this section, we analyse the behaviour of the Daisy Bloom filter with the values $k_x$ denoting the number of hash functions that we use to hash $x$ into the array. Let $X_i$ denote the number of hash functions that are employed when we sample in the $i^{th}$ round, i.e., $X_i = k_x$ with probability $p_x$. Then $X = \sum_i X_i$ denotes the number of locations that are set in the Bloom filter (where the same location might be set multiple times). Moreover, $\mathbb{E}_{\mathcal{P}_n}[X] = n \cdot \sum_{x\in\mathcal{U}} p_x k_x = \mathsf{LB}(\mathcal{P}_n, \mathcal{Q}, \varepsilon)$, since the $\{X_i\}_i$ are identically distributed. We then set the length $m$ of the Daisy Bloom filter array to

$$m \triangleq \mathbb{E}_{\mathcal{P}_n}[X] \,/\, \ln 2 \ .$$

The remainder of this section is dedicated to proving the following statement:

▶ **Theorem 12.** *Assume that $\mathcal{P}_n$ and $\mathcal{Q}$ satisfy $n\sum_{x\in\mathcal{U}} p_x q_x \leq \varepsilon$. Then there exists $\mathcal{T} \subseteq \mathbb{P}(U)$ such that $S \in \mathcal{T}$ with high probability over the randomness of $\mathcal{P}_n$, and for all $S \in \mathcal{T}$ the Daisy Bloom Filter is a $(\mathcal{Q}, \varepsilon)$-filter for $S$. The Daisy Bloom filter uses $\log(e) \cdot LB(\mathcal{P}_n, \mathcal{Q}, \varepsilon) + O(n)$ bits and executes all operations in at most $\lceil \log(1/\varepsilon)\rceil$ time in the worst case.*

The sets in $\mathcal{T}$ are the sets for which $X \approx \mathbb{E}_{\mathcal{P}_n}[X] = \mathsf{LB}(\mathcal{P}_n, \mathcal{Q}, \varepsilon)$. The reason we constrain ourselves to these sets, is that if $X \gg \mathsf{LB}(\mathcal{P}_n, \mathcal{Q}, \varepsilon)$ then most bits will be set to 1 which will make the false positive rate large. We will bound the probability that $X \gg \mathsf{LB}(\mathcal{P}_n, \mathcal{Q}, \varepsilon)$ by using Bernstein's inequality and here, the following observation becomes crucial:

▶ **Observation 13.** *For every $x \in \mathcal{U}$, $k_x \leq \log(1/\varepsilon)$.*

**Proof.** For $x \in \mathcal{U}_0 \cup \mathcal{U}_1$, we have that $k_x = 0$ which is clearly less than $\log(1/\varepsilon)$. For $x \in \mathcal{U}_3$ we have that $k_x = \log(1/\varepsilon)$ and again the statement holds trivially. For $x \in \mathcal{U}_2$, we have that $q_x \leq p_x$ and so $k_x = \log(1/\varepsilon \cdot q_x/p_x) \leq \log(1/\varepsilon)$. For $x \in \mathcal{U}_4$, we have that $p_x > F/n$ and so $k_x = \log(1/(np_x)) < \log(1/\varepsilon)$. ◀

We are now ready to prove that the random variable $X$ is concentrated around its expectation.

▶ **Lemma 14.** *For any $\delta > 0$,*

$$\Pr_{\mathcal{P}_n}[X > (1+\tau)\cdot\mathbb{E}_{\mathcal{P}_n}[X]] \le \exp\left(-\frac{\tau^2 \ln 2}{2(1+\tau/3)}\cdot\frac{m}{\log(1/\varepsilon)}\right)$$

**Proof.** The random variables $\{X_i\}_i$ are independent and $X_i \le b \triangleq \log(1/\varepsilon)$ for all $i$ by Observation 13. We apply Bernstein's inequality [22]:

$$\Pr_{\mathcal{P}_n}[X - \mathbb{E}_{\mathcal{P}_n}[X] > t] \le \exp\left(-\frac{t^2/2}{n\mathrm{Var}_{\mathcal{P}_n}[X_1] + bt/3}\right).$$

Note that $\mathrm{Var}_{\mathcal{P}_n}[X_i] \le \mathbb{E}_{\mathcal{P}_n}\left[X_i^2\right] \le b\cdot\mathbb{E}_{\mathcal{P}_n}[X_i]$. Setting $t = \tau\cdot\mathbb{E}_{\mathcal{P}_n}[X] = \tau n\cdot\mathbb{E}_{\mathcal{P}_n}[X_1]$, we get that

$$\begin{aligned}\Pr_{\mathcal{P}_n}[X > (1+\tau)\mathbb{E}_{\mathcal{P}_n}[X]] &\le \exp\left(-\frac{\tau^2}{2}\cdot\frac{n^2(\mathbb{E}_{\mathcal{P}_n}[X_1])^2}{nb\cdot\mathbb{E}_{\mathcal{P}_n}[X_1] + \tau/3\cdot nb\cdot\mathbb{E}_{\mathcal{P}_n}[X_1]}\right)\\ &= \exp\left(-\frac{\tau^2}{2(1+\tau/3)}\cdot\frac{n\mathbb{E}_{\mathcal{P}_n}[X_1]}{b}\right).\end{aligned}$$

The claim follows by noticing that $n\mathbb{E}_{\mathcal{P}_n}[X_1] = m\ln 2$. ◀

We can now prove that as long as $X \le (1 + 1/(2\log(1/\varepsilon)))\cdot\mathbb{E}_{\mathcal{P}_n}[X]$, the Daisy Bloom filter is a $(\mathcal{Q}, 6\varepsilon)$-filter for $S$. We consider the fraction $\rho$ of entries in the array that are set to 0 after we have inserted the elements of the set. We then show that $\rho$ is close to 1/2 with high probability over the input set and the randomness of the hash functions. Conditioned on this, we then have that the probability that we make a mistake for $x$ is at most $2^{-k_x+1}$. The false positive rate is then derived similarly to that of Lemma 11.

▶ **Lemma 15.** *Assume that $\mathcal{P}_n$ and $\mathcal{Q}$ satisfy $n\sum_{x\in\mathcal{U}} p_x q_x \le \varepsilon$, and that $X \le (1 + 1/(2\log(1/\varepsilon)))\mathbb{E}_{\mathcal{P}_n}[X]$. Then, whp, the Daisy Bloom filter is a $(\mathcal{Q}, 6\varepsilon)$-filter on $S$.*

**Proof.** Let $\rho \in [0,1]$ denote the fraction of entries in the Daisy Bloom filter that are set to 0 after we have inserted the elements of the set. Recall that the random variable $X$ denotes the total number of entries that are set in the Bloom filter, including multiplicities. In the worst case, all the entries to the filter are distinct, and we have $X$ independent chances to set a specific bit to 1. Therefore

$$\mathbb{E}_h[\rho|X] \ge \left(1-\frac{1}{m}\right)^X \approx e^{-X/m} = 2^{-X/\mathbb{E}_{\mathcal{P}_n}[X]}.$$

Moreover, by applying a Chernoff bound for negatively associated random variables, we have that for any $0 < \gamma < 1$,

$$\Pr_A\left[\rho \le (1-\gamma)\cdot\left(1-\frac{1}{m}\right)^X \middle| X\right] \le \exp\left(-m\left(1-\frac{1}{m}\right)^X\cdot\gamma^2/2\right) \tag{2}$$

We now let $B_\delta$ denote the event that $\left(1-\frac{1}{m}\right)^X > (1-\delta)\cdot\frac{1}{2}$ and $B_\gamma$ the event that $\rho > (1-\gamma)\left(1-\frac{1}{m}\right)^X$. We then choose $0 < \delta < 1$ and $0 < \gamma < 1$ such that $B_\delta$ and $B_{\delta'}$ imply that

$$\rho \ge 1 - 2^{1/\log(1/\varepsilon)}\cdot\frac{1}{2}.$$

Moreover, for our choices of $\delta$ and $\gamma$, we have that both $B_\delta$ and $B_\gamma|B_\delta$ occur with high probability.[6] We refer the reader to the full version [7] for $\delta$ and $\gamma$. Conditioned on $B_\delta$ and $B_\gamma$, we get that, for an $x \notin S$, since $k_x \le \log(1/\varepsilon)$,

$$\Pr_A\left[A(S,x) = \mathsf{YES}|B_\delta \wedge B_\gamma\right] = (1-\rho)^{k_x} \le 2^{k_x/b} \cdot 2^{-k_x} \le 2 \cdot 2^{-k_x}$$

We bound the false positive rate on each partition. For $x \in \mathcal{U}_0$, i.e., with $q_x \le Fp_x$ and $k_x = 0$, we can upper bound the false positive rate as such

$$\sum_{x\in\mathcal{U}_0} q_x \cdot \Pr\left[A(S,x) = \mathsf{YES}|B_\delta \wedge B_\gamma\right] \le \sum_{x\in\mathcal{U}_0} q_x \le \sum_{x\in\mathcal{U}_0} p_x \cdot \varepsilon \ .$$

For $x \in \mathcal{U}_1$ with $p_x > 1/n$ and $k_x = 0$, we have the following

$$\sum_{x\in\mathcal{U}_1} q_x \cdot \Pr\left[A(S,x) = \mathsf{YES}|B_\delta \wedge B_\gamma\right] \le \sum_{x\in\mathcal{U}_1} q_x < n \sum_{x\in\mathcal{U}_1} p_x q_x \ .$$

For $x \in \mathcal{U}_2$ with $k_x = \log(1/\varepsilon \cdot q_x/p_x)$, we have the following

$$\begin{aligned}\sum_{x\in\mathcal{U}_2} q_x \cdot \Pr\left[A(S,x) = \mathsf{YES}|B_\delta \wedge B_\gamma\right] &\le \sum_{x\in\mathcal{U}_2} q_x \cdot 2 \cdot 2^{-k_x} = \sum_{x\in\mathcal{U}_2} q_x \cdot 2 \cdot p_x/q_x \cdot \varepsilon \\ &= \sum_{x\in\mathcal{U}_2} p_x \cdot 2\varepsilon \ .\end{aligned}$$

For $x \in \mathcal{U}_3$ with $k_x = \log(1/\varepsilon)$,

$$\sum_{x\in\mathcal{U}_3} q_x \cdot \Pr\left[A(S,x) = \mathsf{YES}|B_\delta \wedge B_\gamma\right] \le \sum_{x\in\mathcal{U}_3} q_x \cdot 2\varepsilon \le 2\varepsilon \ .$$

For $x \in \mathcal{U}_4$ with $k_x = \log(1/(np_x))$,

$$\sum_{x\in\mathcal{U}_4} q_x \cdot \Pr\left[A(S,x) = \mathsf{YES}|B_\delta \wedge B_\gamma\right] \le \sum_{x\in\mathcal{U}_4} q_x \cdot 2np_x = 2n \sum_{x\in\mathcal{U}_4} p_x q_x \ .$$

For the overall false positive rate, note that the total false positive rate in $\mathcal{U}_0$ and $\mathcal{U}_2$ is at most $2\varepsilon$. Similarly for the false positive rate in $\mathcal{U}_3$. For the remaining partitions $\mathcal{U}_1$ and $\mathcal{U}_4$, we have that it is at most

$$2n \sum_{x\in\mathcal{U}_1\cup\mathcal{U}_4} p_x q_x \ .$$

From our assumption, this later term is at most $2\varepsilon$ as well. ◀

Combining the above with Lemma 14 we get that with probability $1 - \exp\left(-\frac{m}{\Theta(\log^3(1/\varepsilon))}\right)$ over the randomness of the input set, the Daisy Bloom filter is a $(\mathcal{Q}, 6\varepsilon)$-filter for $S$. This is exactly the statement of Theorem 12.

[6] We implicitly assume here that $2^{1/\log(1/\varepsilon)}. \le 2$, i.e., $\varepsilon \le 1/2$. Notice that this does not affect the overall result, since the false positive rate we obtain is $5 \cdot \varepsilon$.

## References

1 Yuriy Arbitman, Moni Naor, and Gil Segev. Backyard cuckoo hashing: Constant worst-case operations with a succinct representation. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 787–796. IEEE, 2010. See also arXiv:0912.5424v3.

2 Michael A. Bender, Martin Farach-Colton, Mayank Goswami, Rob Johnson, Samuel McCauley, and Shikha Singh. Bloom filters, adaptivity, and the dictionary problem. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 182–193, 2018. doi:10.1109/FOCS.2018.00026.

3 Michael A Bender, Martin Farach-Colton, Rob Johnson, Bradley C Kuszmaul, Dzejla Medjedovic, Pablo Montes, Pradeep Shetty, Richard P Spillane, and Erez Zadok. Don't thrash: How to cache your hash on flash. In *3rd Workshop on Hot Topics in Storage and File Systems (HotStorage 11)*, 2011.

4 Michael A. Bender, Martin Farach-Colton, John Kuszmaul, William Kuszmaul, and Mingmou Liu. On the optimal time/space tradeoff for hash tables. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20–24, 2022*, pages 1284–1297. ACM, 2022. doi:10.1145/3519935.3519969.

5 Ioana O. Bercea and Guy Even. A dynamic space-efficient filter with constant time operations. In Susanne Albers, editor, *17th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2020, June 22-24, 2020, Tórshavn, Faroe Islands*, volume 162 of *LIPIcs*, pages 11:1–11:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.SWAT.2020.11.

6 Ioana O. Bercea and Guy Even. Dynamic dictionaries for multisets and counting filters with constant time operations. In Anna Lubiw and Mohammad R. Salavatipour, editors, *Algorithms and Data Structures – 17th International Symposium, WADS 2021, Virtual Event, August 9-11, 2021, Proceedings*, volume 12808 of *Lecture Notes in Computer Science*, pages 144–157. Springer, 2021. doi:10.1007/978-3-030-83508-8_11.

7 Ioana O. Bercea, Jakob Bæk Tejs Houen, and Rasmus Pagh. Daisy bloom filters. *CoRR*, abs/2205.14894, 2022. doi:10.48550/arXiv.2205.14894.

8 Ioana Oriana Bercea and Guy Even. An extendable data structure for incremental stable perfect hashing. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20–24, 2022*, pages 1298–1310. ACM, 2022. doi:10.1145/3519935.3520070.

9 Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970. doi:10.1145/362686.362692.

10 Flavio Bonomi, Michael Mitzenmacher, Rina Panigrahy, Sushil Singh, and George Varghese. An improved construction for counting bloom filters. In *Algorithms–ESA 2006: 14th Annual European Symposium, Zurich, Switzerland, September 11-13, 2006. Proceedings 14*, pages 684–695. Springer, 2006.

11 Andrei Z. Broder and Michael Mitzenmacher. Survey: Network applications of bloom filters: A survey. *Internet Math.*, 1(4):485–509, 2003. doi:10.1080/15427951.2004.10129096.

12 Jehoshua Bruck, Jie Gao, and Anxiao Jiang. Weighted bloom filter. In *International Symposium on Information Theory (ISIT)*, pages 2304–2308. IEEE, 2006. doi:10.1109/ISIT.2006.261978.

13 Clément Canonne and Ronitt Rubinfeld. Testing probability distributions underlying aggregated data. In *International Colloquium on Automata, Languages, and Programming*, pages 283–295. Springer, 2014.

14 Xinyuan Cao, Jingbang Chen, Li Chen, Chris Lambert, Richard Peng, and Daniel Sleator. Learning-augmented b-trees, 2023. arXiv:2211.09251.

15 Larry Carter, Robert W. Floyd, John Gill, George Markowsky, and Mark N. Wegman. Exact and approximate membership testers. In Richard J. Lipton, Walter A. Burkhard, Walter J. Savitch, Emily P. Friedman, and Alfred V. Aho, editors, *Proceedings of the 10th Annual ACM Symposium on Theory of Computing, May 1-3, 1978, San Diego, California, USA*, pages 59–65. ACM, 1978.

16 Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *International Colloquium on Automata, Languages, and Programming*, pages 693–703. Springer, 2002.

17 Graham Cormode and Shan Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

18 Zhenwei Dai and Anshumali Shrivastava. Adaptive learned bloom filter (ada-bf): Efficient utilization of the classifier with application to real-time information filtering on the web. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

19 Niv Dayan, Ioana O. Bercea, Pedro Reviriego, and Rasmus Pagh. Infinifilter: Expanding filters to infinity and beyond. *Proc. ACM Manag. Data*, 1(2):140:1–140:27, 2023. `doi:10.1145/3589285`.

20 Martin Dietzfelbinger and Rasmus Pagh. Succinct data structures for retrieval and approximate membership. In *International Colloquium on Automata, Languages, and Programming*, pages 385–396. Springer, 2008.

21 Yihe Dong, Piotr Indyk, Ilya Razenshteyn, and Tal Wagner. Learning space partitions for nearest neighbor search. In *International Conference on Learning Representations (ICLR)*, 2020.

22 Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms.* Cambridge University Press, USA, 2012.

23 Talya Eden, Piotr Indyk, Shyam Narayanan, Ronitt Rubinfeld, Sandeep Silwal, and Tal Wagner. Learning-based support estimation in sublinear time. In *International Conference on Learning Representations*, 2020.

24 Tomer Even, Guy Even, and Adam Morrison. Prefix filter: Practically and theoretically better than bloom. *Proc. VLDB Endow.*, 15(7):1311–1323, 2022. `doi:10.14778/3523210.3523211`.

25 Bin Fan, Dave G Andersen, Michael Kaminsky, and Michael D Mitzenmacher. Cuckoo filter: Practically better than bloom. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, pages 75–88, 2014.

26 Li Fan, Pei Cao, Jussara Almeida, and Andrei Z Broder. Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM transactions on networking*, 8(3):281–293, 2000.

27 Paolo Ferragina, Hans-Peter Lehmann, Peter Sanders, and Giorgio Vinciguerra. Learned monotone minimal perfect hashing. In Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman, editors, *31st Annual European Symposium on Algorithms, ESA 2023, September 4-6, 2023, Amsterdam, The Netherlands*, volume 274 of *LIPIcs*, pages 46:1–46:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. `doi:10.4230/LIPICS.ESA.2023.46`.

28 Paolo Ferragina, Fabrizio Lillo, and Giorgio Vinciguerra. Why are learned indexes so effective? In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research.* PMLR, 2020.

29 Paolo Ferragina and Giorgio Vinciguerra. Learned data structures. In Luca Oneto, Nicolò Navarin, Alessandro Sperduti, and Davide Anguita, editors, *Recent Trends in Learning From Data – Tutorials from the INNS Big Data and Deep Learning Conference (INNSBDDL 2019)*, volume 896 of *Studies in Computational Intelligence*, pages 5–41. Springer, 2019. `doi:10.1007/978-3-030-43883-8_2`.

30 Paolo Ferragina and Giorgio Vinciguerra. The pgm-index: a fully-dynamic compressed learned index with provable worst-case bounds. *Proceedings of the VLDB Endowment*, 13(8):1162–1175, 2020.

31 Alex Galakatos, Michael Markovitch, Carsten Binnig, Rodrigo Fonseca, and Tim Kraska. Fiting-tree: A data-aware index structure. In *Proceedings International Conference on Management of Data (SIGMOD)*, pages 1189–1206, 2019.

**32** Chen-Yu Hsu, Piotr Indyk, Dina Katabi, and Ali Vakilian. Learning-based frequency estimation algorithms. In *International Conference on Learning Representations*, 2019.

**33** Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In Gautam Das, Christopher M. Jermaine, and Philip A. Bernstein, editors, *Proceedings of the 2018 International Conference on Management of Data (SIGMOD)*, pages 489–504. ACM, 2018.

**34** Mingmou Liu, Yitong Yin, and Huacheng Yu. Succinct filters for sets of unknown sizes. *arXiv preprint*, 2020. `arXiv:2004.12465`.

**35** Lailong Luo, Deke Guo, Richard T. B. Ma, Ori Rottenstreich, and Xueshan Luo. Optimizing bloom filter: Challenges, solutions, and comparisons. *IEEE Commun. Surv. Tutorials*, 21(2):1912–1949, 2019. `doi:10.1109/COMST.2018.2889329`.

**36** Samuel McCauley, Benjamin Moseley, Aidin Niaparast, and Shikha Singh. Online list labeling with predictions, 2023. `arXiv:2305.10536`.

**37** Michael Mitzenmacher. A model for learned bloom filters and optimizing by sandwiching. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

**38** Michael Mitzenmacher, Salvatore Pontarelli, and Pedro Reviriego. Adaptive cuckoo filters. In *2018 Proceedings of the Twentieth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 36–47. SIAM, 2018.

**39** Moni Naor and Noa Oved. Bet-or-pass: Adversarially robust bloom filters. In *Theory of Cryptography Conference*, pages 777–808. Springer, 2022.

**40** Moni Naor and Eylon Yogev. Bloom filters in adversarial environments. In *Annual Cryptology Conference*, pages 565–584. Springer, 2015.

**41** Anna Pagh, Rasmus Pagh, and S. Srinivasa Rao. An optimal Bloom filter replacement. In *SODA*, pages 823–829. SIAM, 2005.

**42** Rasmus Pagh, Gil Segev, and Udi Wieder. How to approximate a set without knowing its size in advance. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 80–89. IEEE, 2013.

**43** Prashant Pandey, Michael A Bender, Rob Johnson, and Rob Patro. A general-purpose counting filter: Making every bit count. In *Proceedings of the 2017 ACM international conference on Management of Data*, pages 775–787, 2017.

**44** Prashant Pandey, Alex Conway, Joe Durie, Michael A. Bender, Martin Farach-Colton, and Rob Johnson. Vector quotient filters: Overcoming the time/space trade-off in filter design. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, pages 1386–1399. ACM, 2021. `doi:10.1145/3448016.3452841`.

**45** Ely Porat. An optimal Bloom filter replacement based on matrix solving. In *International Computer Science Symposium in Russia*, pages 263–273. Springer, 2009.

**46** Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ml predictions. *Advances in Neural Information Processing Systems (NeurIPS)*, 31, 2018.

**47** MTCAJ Thomas and A Thomas Joy. *Elements of information theory*. Wiley-Interscience, 2006.

**48** Kapil Vaidya, Eric Knorr, Michael Mitzenmacher, and Tim Kraska. Partitioned learned bloom filters. In *9th International Conference on Learning Representations (ICLR)*. OpenReview.net, 2021.

**49** Xiujun Wang, Yusheng Ji, Zhe Dang, Xiao Zheng, and Baohua Zhao. Improved weighted bloom filter and space lower bound analysis of algorithms for approximated membership querying. In *Database Systems for Advanced Applications (DASFAA)*, volume 9050 of *Lecture Notes in Computer Science*, pages 346–362. Springer, 2015. `doi:10.1007/978-3-319-18123-3_21`.