A Space-Efficient Dynamic Dictionary for Multisets with Constant Time Operations [*]

Ioana O. Bercea[†]   Guy Even[†]

Abstract

We consider the dynamic dictionary problem for multisets. Given an upper bound n on the total cardinality of the multiset (i.e., including multiplicities) at any point in time, the goal is to design a data structure that supports multiplicity queries and allows insertions and deletions to the multiset (i.e., the dynamic setting). The data structure must be space-efficient (the space is $1 + o(1)$ times the information-theoretic lower bound) and support all operations in constant time with high probability.

In this paper, we present the first dynamic dictionary for multisets that achieves these performance guarantees. This answers an open problem of Arbitman, Naor and Segev [ANS10]. The previously best-known construction of Pagh, Pagh and Rao [PPR05] supports membership in constant time, multiplicity queries in $O(\log n)$ time in the worst case, and insertions and deletions in constant expected amortized time. The main technical component of our solution is a strategy for efficiently storing variable-length binary counters using weighted balls-into-bins experiments in which balls have logarithmic weights.

We also obtain a counting filter that approximates multiplicity queries with a one sided error, using the reduction of Carter *et al.* [CFG+78]. Counting filters have received significant attention over the years due to their applicability in practice. We present the first counting filter with constant time operations.

1 Introduction

We consider the dynamic dictionary problem for multisets. The special case in which every element of the universe can appear at most once is a fundamental problem in data structures and has been well studied [ANS10, PPR05, RR03, DadHPP06]. In the case of multisets, elements can have arbitrary multiplicities and we are given an upper bound n on the total cardinality of the multiset (i.e., including multiplicities) at any point in time. The goal is to design a data structure that supports multiplicity queries and allows insertions and deletions to the multiset (i.e., the dynamic setting).

A related problem is that of supporting *approximate* membership and multiplicity queries. The classic approximate setting allows one-sided errors in the form of false positives: given an error parameter $\varepsilon > 0$, the probability of returning a "yes" on an element not in the set must be upper bounded by ε. Such data structures are known as *filters*. For multisets, the corresponding data structure is known as a *counting* filter. A counting filter returns a count that is at least the multiplicity of the element in the multiset and overcounts with probability bounded by ε. Counting filters have received significant attention over the years due to their applicability in

---

[†]Tel Aviv University, Tel Aviv, Israel. Email: `ioana@cs.umd.edu, guy@eng.tau.ac.il`.

practice [FCAB00, CM03, BMP+06]. One of the main applications of dictionaries for multisets is precisely in designing counting filters. Namely, Carter *et al.* [CFG+78] showed that by hashing each element into a random fingerprint, one can reduce a counting filter to a dictionary for multisets by storing the fingerprints in the dictionary.

For the design of both dictionaries and filters, the performance measures of interest are the space the data structure takes and the time it takes to perform the operations. For dictionaries, we would like to get close to the lower bound of $\log \binom{u+n}{n} = n \log(u/n) + \Theta(n)$ bits, where u is the size of the universe.[1][2] In the case of filters, the lower bound is at least $n \log(1/\varepsilon) + \Theta(n)$ bits [LP10]. A data structure is *space-efficient* if the total number of bits it requires is within $(1 + o(1))$ of the lower bound, where the $o(1)$ term converges to zero as n tends to infinity. The goal is to design data structures that are space-efficient with high probability.[3] We would like to support queries, insertions and deletions in constant time in the word RAM model. The constant time guarantees should be in the worst case with high probability (see [BM01, KM07, ANS09, ANS10] for a discussion on the shortcomings of expected or amortized performance in practical scenarios). We assume that each memory access can read/write a word of $w = \log n$ contiguous bits.

The current best known dynamic dictionary for multisets was designed by Pagh, Pagh, Rao [PPR05] based on the dictionary for sets of Raman and Rao [RR03]. The dictionary is space-efficient and supports membership queries in constant time in the worst case. Insertions and deletions take amortized expected constant time and multiplicity queries take $O(\log n)$ in the worst case. In the case of sets, the state-of-the-art dynamic dictionary of Arbitman, Naor and Segev [ANS10] achieves the "best of both worlds": it is space-efficient and supports all operations in constant time whp. Arbitman *et al.* [ANS10] leave it as an open problem whether a similar result can be achieved for multisets.

Recently, progress on this problem was achieved by Bercea and Even [BE20] who designed a constant-time dynamic space-efficient dictionary for *random* multisets. In a random multiset, each element is sampled independently and uniformly at random from the universe. In this paper, we build upon their work and present the first space-efficient dynamic dictionary for (arbitrary) multisets with constant time operations in the worst case with high probability, resolving the question of Arbitman *et al.* [ANS10] in the positive. We also obtain a counting filter with similar guarantees.

1.1 Results

In the following theorem, we assume that the size of the universe $\mathcal{U}$ is polynomial in n.[4] *Overflow* refers to the event that the space allocated in advance for the dictionary does not suffice.

Theorem 1 (dynamic multiset dictionary). *There exists a dynamic dictionary that maintains a multiset of cardinality at most n from the universe $\mathcal{U} = \{0, 1\}^{\log_2 u}$ with the following guarantees: (1) For every polynomial in n sequence of operations (multiplicity query, insertion, deletion), the dictionary does not overflow whp. (2) If the dictionary does not overflow, then every operation can be completed in constant time. (3) The required space is $(1 + o(1)) \cdot n \log(u/n) + O(n)$ bits.*

Our dictionary construction considers a natural separation into the *sparse* and *dense* case based on the size of the universe relative to n. The sparse case, defined when $\log(u/n) = \omega(\log \log n)$,

---

[1]All logarithms are base 2 unless otherwise stated. $\ln x$ is used to denote the natural logarithm.

[2] This equality holds when u is significantly larger than n.

[3]By with high probability (whp), we mean with probability at least $1 - 1/n^{\Omega(1)}$. The constant in the exponent can be controlled by the designer and only affects the $o(1)$ term in the space of the dictionary or the filter.

[4]This is justified by mapping $\mathcal{U}$ to [poly(n)] using 2-independent hash functions [DadHPP06].

presents a more straightforward challenge for dictionary design because the dictionary construction can afford to store additional $\Theta(\log \log n)$ bits per element without sacrificing space-efficiency. In this case, the dictionary for multisets is based on a simple observation. Namely, elements with multiplicity at most $\log^3 n$ can be stored in a space-efficient dictionary for sets by attaching to each element a fixed-length counter of $3 \log \log n$ bits (see Section 3).

The majority of the paper is focused on designing a dictionary for multisets in the *dense* case, in which $\log(u/n) = O(\log \log n)$.[5] Following [BE20], we hash distinct elements into a first level that consists of small space-efficient "bin dictionaries" of fixed capacity. The first level only stores elements of multiplicity strictly smaller than $\log^3 n$, just like in the dense case. However, we employ variable-length counters to encode multiplicities and store them in a separate structure called a "counter dictionary". We allocate one counter dictionary per each bin dictionary. The capacity of a counter dictionary is an upper bound on the total length of the counters it stores and is linear in the capacity of the associated bin dictionary.

Elements that do not fit in the first level are stored in a secondary data structure called the spare. The spare is small enough that it can allocate $\log n$ bit counters for the elements it stores. To bound the number of elements that are stored in the "spare", we cast the process of hashing counters into counter dictionaries as a weighted balls-into-bins experiment in which balls have logarithmic weights (see Sec. 4.4).

As a corollary of Thm. 1, we obtain a counting filter with the following guarantees.[6]

Corollary 2 (dynamic counting filter). *There exists a dynamic counting filter for multisets of cardinality at most n from a universe $\mathcal{U} = \{0,1\}^u$ such that the following hold: (1) For every polynomial in n sequence of operations (multiplicity query, insertion, deletion), the filter does not overflow whp. (2) If the filter does not overflow, then every operation can be completed in constant time. (3) The required space is $(1 + o(1)) \cdot \log(1/\varepsilon) \cdot n + O(n)$ bits. (4) For every count query, the probability of overcounting is bounded by ε.*

1.2 Related Work

The dictionary for multisets of Pagh *et al.* [PPR05] is space-efficient and supports membership queries in constant time in the worst case. Insertions and deletions take amortized expected constant time and multiplicity queries take $O(\log c)$ for a multiplicity of c. Multiplicities are represented "implicitly" by a binary counter whose operations (query, increment, decrement) are simulated as queries and updates to dictionaries on sets.[7] Increments and decrements to the counter take $O(1)$ bit probes (and hence $O(1)$ dictionary operations) but decoding the multiplicity takes $O(\log n)$ time in the worst case. We are not aware of any other dictionary constructions for multisets.[8]

Dynamic dictionaries for sets have been extensively studied [DadH90, DDMM05, DadHPP06, RR03, FPSS05, Pan05, DW07, ANS09, ANS10]. The dynamic dictionary for sets of Arbitman *et al.* [ANS10] is space-efficient and supports operations in constant time whp. Their construction cannot be generalized in a straightforward manner to handle multisets. Specifically, their dictionary

---

[5]This case is especially relevant in the approximate membership setting in which we have $u/n = 1/\varepsilon$ due to the reduction of Carter *et al.* [CFG$^+$78]. In this setting, the dense case arises in applications in which n is large and the error probability ε is a constant (say $\varepsilon = 1\%$).

[6]Note that we allow ε to be as small as $n/|\mathcal{U}|$ (below this threshold, simply use a dictionary).

[7]To be more exact, for each bit of the counter, the construction in Pagh *et al.* [PPR05] allocates a dictionary on sets such that the value of the bit can be retrieved by performing a lookup in the dictionary. Updating a bit of the counter is done by inserting or deleting elements in the associated dictionary.

[8]Data structures for predecessor and successor queries such as [PT14] can support multisets but they do not meet the required performance guarantees for the special case of (just) supporting multiplicity queries.

maintains a spare of size $\Omega\left(\frac{\log\log n}{(\log n)^{1/3}} \cdot n\right)$ elements and hence, cannot store counters of length $\log n$ per element. In contrast, the spare in our construction is guaranteed to store at most $3n/(\log^3 n)$ elements whp.

In terms of counting filters, several constructions do not come with worst case guarantees for storing arbitrary multisets [FCAB00, BMP+06]. The only previous counting filter with worst case guarantees we are aware of is the Spectral Bloom filter of Cohen and Matias [CM03] (with over 450 citations in Google Scholar). The construction is a generalization of the Bloom filter and hence requires $\Theta(\log(1/\varepsilon))$ memory accesses per operation. The space usage is similar to that of a Bloom filter and depends on the sum of logs of multiplicities. Consequently, when the multiset is a set, the leading constant is 1.44, and hence Spectral Bloom Filters are not space-efficient in general.

1.3 Paper Organization

Preliminaries are in Sec. 2. The construction for the sparse case can be found in Sec. 3 and the one for the dense case is described and analyzed in Sec. 4. Section 5 describes how our analysis works without the assumption of access to truly random hash functions. Corollary 2 is proved in Sec. 6. Appendix A reviews standard implementation techniques.

2 Preliminaries

2.1 Notation and Definitions

For $k > 0$, let $[k]$ denote the set $\{0, \ldots, \lceil k \rceil - 1\}$. For a string $a \in \{0,1\}^*$, let $|a|$ denote the length of a in bits. We often abuse notation, and regard elements in $[k]$ as binary strings of length $\log k$. Let $\mathcal{U} = \{0,1\}^{\log u}$ denote the universe of all possible elements.

Definition 3 (multiset). *A multiset $\mathcal{M}$ over $\mathcal{U}$ is a function $\mathcal{M} : \mathcal{U} \to \mathbb{N}$. We refer to $\mathcal{M}(x)$ as the* multiplicity *of x.*

The cardinality *of a multiset $\mathcal{M}$ is denoted by $|\mathcal{M}|$ and defined by $|\mathcal{M}| \triangleq \sum_{x \in \mathcal{U}} \mathcal{M}(x)$. The* support *of the multiset is denoted by $\sigma(\mathcal{M})$ and is defined by $\sigma(\mathcal{M}) \triangleq \{x \mid \mathcal{M}(x) > 0\}$.*

Operations over Dynamic Multisets. We consider the following operations: $\mathsf{insert}(x)$, $\mathsf{delete}(x)$, and $\mathsf{count}(x)$. Let $\mathcal{M}_t$ denote the multiset after t operations. A *dynamic multiset* $\{\mathcal{M}_t\}_t$ is specified by a sequence $\{\mathsf{op}_t\}_{t \geq 1}$ of as follows.[9]

$$\mathcal{M}_t(x) \triangleq \begin{cases} 0 & \text{if } t = 0 \\ \mathcal{M}_{t-1}(x) + 1 & \text{if } \mathsf{op}_t = \mathsf{insert}(x) \\ \mathcal{M}_{t-1}(x) - 1 & \text{if } \mathsf{op}_t = \mathsf{delete}(x) \\ \mathcal{M}_{t-1}(x) & \text{otherwise.} \end{cases}$$

We say that a dynamic multiset $\{\mathcal{M}_t\}_t$ has cardinality at most n if $|\mathcal{M}_t| \leq n$, for every t.

Dynamic Dictionary for Multisets. A *dynamic dictionary for multisets* maintains a dynamic multiset $\{\mathcal{M}_t\}_t$. The response to $\mathsf{count}(x)$ is simply $\mathcal{M}_t(x)$.

---

[9]We require that $\mathsf{op}_t = \mathsf{delete}(x_t)$ only if $\mathcal{M}_{t-1}(x_t) > 0$, i.e. if x is not in the multiset, then a delete operation does not make its multiplicity negative.

Dynamic Counting Filter. A *dynamic counting filter* maintains a dynamic multiset $\{\mathcal{M}_t\}_t$ and is parameterized by an error parameter $\varepsilon \in (0,1)$. Let out_t denote the response to a $\mathsf{count}(x_t)$ at time t. We require that the output out_t satisfy the following conditions:

$$\mathsf{out}_t \geq \mathcal{M}_t(x_t) \tag{1}$$

$$\Pr\left[\mathsf{out}_t > \mathcal{M}_t(x_t)\right] \leq \varepsilon . \tag{2}$$

Namely, out_t is an approximation of $\mathcal{M}_t(x_t)$ with a one-sided error.

Definition 4 (overcounting)**.** *Let Err_t denote the event that $\mathsf{op}_t = \mathsf{count}(x_t)$, and $\mathsf{out}_t > \mathcal{M}_t(x_t)$.*

Note that overcounting generalizes false positive events in filters over sets. Indeed, a false positive event occurs if $\mathcal{M}_t(x_t) = 0$ and $\mathsf{out}_t > 0$.[10]

2.2 The Model

Memory Access Model. We assume that the data structures are implemented in the RAM model in which the basic unit of one memory access is a word. Let w denote the memory word length in bits. We assume that $w = \Theta(\log n)$. See Appendix A for a discussion on how the computations we perform over words are implemented in constant time.

Success Probability. We prove that overflow occurs with probability at most $1/\mathsf{poly}(n)$ and that one can control the degree of the polynomial (the degree of the polynomial only affects the $o(1)$ term in the size bound). The probability of an overflow depends only on the random choices that the dictionary makes.

Hash Functions. Our dictionary uses the succinct hash functions of Arbitman *et al.* [ANS10] which have a small representation and can be evaluated in constant time. For simplicity, we first analyze the data structure assuming fully random hash functions (Sec. 4.4). In Sec. 5, we prove that the same arguments hold when we use succinct hash functions. The filter reduction additionally employs pairwise independent hash functions.

3 Dictionary for Multisets via Dictionary+Retrieval (Sparse Case)

In this section, we show how to design a multiset dictionary using any dictionary on sets that supports attaching satellite data of $O(\log n)$ bits per element. Such a dictionary with satellite data supports the operations: query, insert, delete, retrieve, and update. A retrieve operation for x returns the satellite data of x. An update operation for x with new satellite data d stores d as the new satellite data of x. The reduction incurs a penalty of $\Theta(\log \log n)$ extra bits per element. Hence, a space-efficient multiset dictionary is obtained from a space-efficient dictionary only if $\log(u/n) = \omega(\log \log n)$.

Let $\mathsf{Dict}(n, r)$ denote a dynamic dictionary for sets of cardinality at most n, where r bits of satellite data are attached to each element. Let $\mathsf{MS\text{-}Dict}(n)$ denote a dynamic dictionary for multisets of cardinality at most n.

The reduction is summarized in the following observation.

---

[10] The probability space is induced only by the random choices (i.e., choice of hash functions) that the filter makes. Note also that if $\mathsf{op}_t = \mathsf{op}_{t'} = \mathsf{count}(x)$, then the events Err_t and $\mathrm{Err}_{t'}$ need not be independent.

Observation 5. *One can implement* MS-Dict(n) *using two dynamic dictionaries:* $D_1 = $ Dict$(n, 3 \log \log n)$ *and* $D_2 = $ Dict$(n/(\log^3 n), \log n)$. *Each operation over* MS-Dict *can be performed using a constant number of operations over* D_1 *and* D_2.

Proof Sketch. An element is *light* if its multiplicity is at most $\log^2 n$, otherwise it is *heavy*. Dictionary D_1 is used for storing the light elements, whereas dictionary D_2 is used for storing the heavy elements. The satellite data in both dictionaries is a binary counter of the multiplicity. $\square$

Claim 6. *If* $\log(u/n) = \omega(\log \log n)$, *then there exists a dynamic multiset dictionary that is space-efficient and supports operations in constant time in the worst case whp.*

Proof. A space-efficient implementation of Dict(n, r) (for $r = O(\log n)$) with constant time per operation can be obtained from the dictionary of Arbitman *et al.* [ANS10] (see also [BE20]). The space of such a dictionary is $(1 + o(1)) \cdot (\log(u/n) + r) \cdot n + O(n)$ bits. Instantiating this space for D_1 and D_2 from Observation 5 yields a multiset dictionary MS-Dict(n) with space: $(1 + o(1)) \cdot ((\log(u/n) + 3 \log \log n) \cdot n + O(n)$. In the sparse case $\log(u/n) = \omega(\log \log n)$, and hence the obtained MS-Dict(n) is space efficient. $\square$

This completes the proof of Theorem 1 for the sparse case.

Remark. An alternative solution stores the multiplicities in an array separately from a dictionary that stores the support of the multiset. Let s denote the cardinality of the support of the multiset. Let $h : \mathcal{U} \to [s + o(s)]$ be a dynamic perfect hashing that requires $\Theta(s \log \log s)$ bits and supports operations in constant time (such as the one in [DadHPP06]). Store the (variable-length) binary counter for x at index $h(x)$ in the array. The array can be implemented in space that is linear in the total length of the counters and supports query and update operations in constant time [BB08].

4 Dictionary for Multisets (Dense Case)

In this section, we prove Theorem 1 for the case in which $\log(u/n) = O(\log \log n)$, which we call the *dense* case. We refer to this dictionary construction as the *MS-Dictionary* (Multiset Dictionary) in the dense case.

The MS-Dictionary construction follows the same general structure as in [ANS10, DadHPP06, BE20]. Specifically, it consists of two levels of dictionaries. The first level is designed to store the majority of the elements (Sec. 4.2). An element is stored in the first level provided that its multiplicity is at most $\log^3 n$ and there is enough capacity. Otherwise, the element is stored in the second level, which is called the *spare* (Sec. 4.3).

The first level of the MS-Dictionary consists of m *bin dictionaries* $\{\mathsf{BD}_i\}_{i \in [m]}$ together with m *counter dictionaries* $\{\mathsf{CD}_i\}_{i \in [m]}$. Each bin dictionary can store at most $n_B = (1 + \delta)B$ distinct elements, where $\delta = o(1)$ and $B \triangleq n/m$ denotes the mean occupancy of each bin dictionary. We say that a bin dictionary is *full* if it stores n_B elements in it.

Each counter dictionary stores variable-length binary counters. Each counter represents the multiplicity of an element in the associated bin dictionary. Each counter dictionary can store counters whose total length in bits is at most $6B$. We say that a counter dictionary is *full* if the total length of the counters stored in it is $6B$ bits.

Elements with high multiplicity or whose BD or CD are full are stored in the spare, as formulated in the following invariant:

Invariant 7. *An element x such that $\mathcal{M}_t(x) > 0$ is stored in the spare at time t if: (1) $\mathcal{M}_t(x) \geq \log^3 n$, (2) the bin dictionary corresponding to x is full, or (3) the counter dictionary corresponding to x is full.*

We denote the upper bound on the cardinality of the support of the multiset stored in the spare by n_S (the value of n_s is specified later). We say that the spare *overflows* when more than n_S elements are stored in it.

4.1 Hash Functions

We employ a permutation $\pi : \mathcal{U} \to \mathcal{U}$. We define $h^b : \mathcal{U} \to [m]$ to be the leftmost $\log m$ bits of the binary representation of $\pi(x)$ and by $h^r : \mathcal{U} \to [u/m]$ to be the remaining $\log(u/m)$ bits of x. An element x is hashed to the bin dictionary of index $h^b(x)$. Hence storing x in the first level of the dictionary amounts to storing $h^r(x)$ in BD_i, where $i = h^b(x)$, and storing $\mathcal{M}_t(x)$ in CD_i. (This reduction in the universe size is often called "quotienting" [Knu73, Pag01, PPR05, DadHPP06]).

The overflow analysis in Sec. 4.4 assume truly random permutations. In Sec. 5, we discuss how one can replace this assumption with the succinct hash functions of Arbitman *et al.* [ANS10].

4.2 The First Level of the Dictionary

We follow the same parametrization as in [BE20]. Namely, we set the average occupancy of a bin dictionary to be $B \triangleq (\log n)/\log(u/n)$ and set $\delta \triangleq \Theta(\frac{\log \log n}{\sqrt{B}})$.

Bin Dictionaries. Each bin dictionary (BD) is a deterministic dictionary for sets of cardinality at most n_B that supports queries, insertions and deletions. The implementation of a bin dictionary using global lookup tables [ANS10] or Elias-Fano encoding [BE20] is briefly reviewed in Appendix A. We remark that each BD is space-efficient, meaning it requires $n_B \cdot \log(u/n) + O(n_B)$ bits. Moreover, each BD fits in a constant number of words and performs queries, insertions and deletions in constant time.

Counter Dictionaries. Each counter dictionary CD_i stores a vector of multiplicities of the elements stored in the corresponding bin dictionary BD_i. The order of the multiplicities stored in CD_i is the same order in which the corresponding elements are stored in BD_i. Multiplicities in CD_i are stored by variable-length counters. We employ a trivial 2-bit alphabet to encode $0, 1$ and "end-of-counter" symbols for encoding the multiplicities. Hence, the length of a counter c is $\lceil \log_2 c \rceil$ bits and its encoding $2(1 + \lceil \log_2 c \rceil)$ bits long. The contents of CD_i is simply a concatenation of the encoding of the counters. We allocate $2(6B + n_B)$ bits per CD.[11]

The CD supports the operations of multiplicity query, increment and decrement. These operations are carried out naturally in constant time because each CD_i fits in a word. We note that an increment may cause the CD to be full, in which case x is deleted from the bin dictionary and is inserted to the spare together with its updated counter. Similarly, a decrement may zero the counter, in which case x is deleted from the bin dictionary (and hence its multiplicity is also deleted from the counter dictionary).

4.3 The Spare

The spare is a high performance space-inefficient dictionary for multisets. It stores at most $n_S = O(n/\log^3 n)$ distinct elements. Each element stored in the spare can have a multiplicity as high

---

[11]Note, however, that we define a CD to be full if the sum of counter lengths is $6B$ (even if we did not use all its space). The justification for this definition is to simplify the analysis.

as n. It supports all the operations of the dictionary in constant time. In addition, the spare also moves elements back to the first level if their insertion no longer violates Invariant 7.

We propose to implement the spare using the dynamic dictionary of Arbitman *et al.* [ANS09] in which we append $\log n$-bit counters to each element. We briefly review the construction here. The dictionary is a de-amortized construction of the cuckoo hash table of Pagh and Rodler [PR01]. Namely, each element is assigned two locations in an array. If upon insertion, both locations are occupied, then space for the new element is made by "relocating" an element occupying one of the two locations. Long chains of relocations are "postponed" by employing a queue of pending insertions. Thus, each operation is guaranteed to perform in constant time in the worst case. The space that the dictionary occupies is $O(n_S \log(u/n)) + O(n)$. The counters increase the space of the spare by $O(n_S \log n) = o(n)$ bits.

The construction in [ANS09] is used as a spare in the space-efficient dynamic filter in [ANS10]. We use it a similar manner to maintain Invariant 7 in a "lazy" fashion. Namely, if an element x residing in the spare is no longer in violation of Invariant 7 (for instance, due to a deletion in the bin dictionary), we do not immediately move x from the spare back to its bin dictionary. Instead, we "delay" such an operation until x is examined during a chain of relocations. Specifically, during an insertion to the spare, for each evicted element, one checks if this element is still in violation of Invariant 7. If it is not, then it is deleted from the spare and inserted into the first level. This increases the time it takes to perform an insertion to the spare only by a constant. Moreover, it does not affect the overflow probability of the spare.

4.4 Overflow Analysis

The event of an overflow occurs if more than n_S distinct elements are stored in the spare. In this section, we prove that overflow does not occur whp with respect to perfectly random hash functions. In Sec. 5, we discuss how this analysis can be modified when we employ succinct hash functions.

The analysis proceeds in two stages. First, we consider the incremental setting (in which elements of the multiset are inserted one-by-one and there are no deletions). We prove that overflow does not occur whp if $n_S = 3n/\log^3(n)$. The proof for the dynamic setting (deletions and insertions) is based on Invariant 7. Namely, Invariant 7 reduces the dynamic setting to an incremental setting. Formally, the probability of overflow at time t (after a sequence of deletions and insertions) equals the probability of an overflow had the elements of $\mathcal{M}_t$ been inserted one-by-one (no deletions). Hence, overflow does not occur whp over a polynomial number of operations in the dynamic setting by applying a union bound.

Recall that each component of the first level of the dictionary has capacity parameters: each bin dictionary has an upper bound of $n_B = (1 + \delta)B$ on the number of distinct elements it stores and each counter dictionary has an upper bound of $6B$ on the total length of the counters it stores. Additionally, the first level only stores elements whose multiplicity is strictly smaller than $\log^3 n$. According to Invariant 7, if the insertion of some element x exceeds these bounds, then x is moved to the spare.

We bound the number of elements that go to the spare due to failing one of the conditions of Invariant 7 separately. The number of elements whose multiplicity is at least $\log^3 n$ is at most $n/\log^3 n$. The number of distinct elements that are stored in the spare because their bin dictionary is full is at most $n/\log^3 n$ whp. The proof of this bound can be derived by modifying the proof of Claim 8 (see also [ANS10]). We focus on the number of distinct elements whose counter dictionary is full.

Claim 8. *The number of distinct elements whose corresponding CD is full is at most $n/\log^3 n$ whp.*

Proof. Recall that there are $m = n/B$ counter dictionaries and that each CD stores the multiplicities of at most $n_B = (1+\delta)B$ distinct elements of multiplicity strictly smaller than $\log^3 n$. In a full CD, the sum of the counter lengths reaches $6B$. We start by bounding the probability that the total length of the counters in a CD is at least $6B$.

Formally, consider a multiset $\mathcal{M}$ of cardinality n consisting of s distinct elements $\{x_i\}_{i\in[s]}$ with multiplicities $\{f_i\}_{i\in[s]}$ (note that $\sum_{i\in[s]} f_i = n$). The length of the counter for multiplicity f_i is $w_i \triangleq \lceil \log(f_i+1) \rceil$ (we refer to this quantity as *weight*). For $\beta \in [m]$, let $\mathcal{M}^\beta$ denote the sub-multiset of $\mathcal{M}$ consisting of the elements x_i such that $h^b(x_i) = \beta$. Let C_β denote the event that the weight of $\mathcal{M}^\beta$ is at least $6B$, namely $\sum_{x_i \in \mathcal{M}^\beta} w_i \geq 6B$. We begin by bounding the probability of event C_β occurring.

For $i \in [s]$, define the random variable $X_i \in \{0, w_i\}$, where $X_i = w_i$ if $h^b(x_i) = \beta$ and 0 otherwise. Since the values $\{(h^b(x_i), h^q(x_i))\}_i$ were sampled at random without replacement (i.e., obtained from a random permutation), the random variables $\{X_i\}_i$ are negatively associated. Let $\mu \triangleq \frac{1}{m} \cdot \sum_{i\in[s]} w_i$ denote the expected weight per CD. Clearly, $\mu \leq \frac{n}{m} = B$. We now scale the RVs so that they are in the range $[0,1]$. Since the multiplicities of elements in the first level is strictly smaller than $\log^3 n$, we have that $w_i \leq \log \log^3 n$ (we omit the ceiling to improve readability). We then define $\tilde{X}_i \triangleq X_i / \log \log^3 n$ and $\tilde{\mu} = \mu / \log \log^3 n \leq B / \log \log^3 n$. Then, by Chernoff's bound:

$$
\begin{aligned}
\Pr[C_\beta] &= \Pr\left[\sum_{i\in[s]} X_i \geq 6B\right] \\
&= \Pr\left[\sum_{i\in[s]} \tilde{X}_i \geq \frac{6B}{\log \log^3 n}\right] \\
&\leq 2^{-\frac{6B}{\log \log^3 n}} \\
&= 1/(\log n)^{\omega(1)} .
\end{aligned}
$$

Let $I(C_\beta)$ denote the indicator variable for event C_β. Then $\mathbb{E}\left[\sum_\beta I(C_\beta)\right] \leq n/(\log n)^{\omega(1)}$. Moreover, the RVs $\{I(C_\beta)\}_\beta$ are negatively associated (more weight in bin b implies less weight in bin b'). By Chernoff's bound:

$$
\Pr\left[\sum_b I(C_\beta) \geq \frac{n}{\log^5 n}\right] \leq O(2^{-n/(\log^5 n)}) .
$$

Whp, a bin is assigned at most $\log^2 n$ elements. We conclude that the number of elements that are stored in the spare due to events $\bigcup_b C_\beta$ is at most $n/(\log^3 n)$ whp. $\square$

4.5 Space Analysis

Each bin dictionary takes $n_B \log(u/n) + \Theta(n_B)$ bits, where $n_B = (1+\delta)B$, $B = n/m$ and $\delta = o(1)$. Each CD occupies $\Theta(B)$ bits. Therefore, the first level of the MS-Dictionary takes $(1+\delta)n \log(u/n) + \Theta(n)$ bits. The spare takes $O(n_S \log(u/n)) = o(n)$ bits, since $n_S = \Theta(n/\log^3 n)$. Therefore, the space the whole dictionary takes is $(1 + o(1)) \cdot \log(u/n) + \Theta(n)$ bits. This completes the proof of Theorem 1 for the dense case.

5 Succinct Hash Functions

In this section, we discuss how to replace the assumption of truly random permutations with succinct hash functions (i.e., representation requires $o(n)$ bits) that have constant evaluation time in the RAM model.

We follow the construction in [ANS10], which we describe as follows. Partition the universe into $M = n^{9/10}$ parts using a one-round Feistel permutation (described below) such that the number of elements in each part is at most $n^{1/10} + n^{3/40}$ whp. The permutation uses highly independent hash functions [Sie04,DR09]. Apply the dictionary construction separately in each part with an upper bound of $n^{9/10} + n^{3/40}$ on the cardinality of the set. Within each part, the dictionary employs a k-wise δ-dependent permutation. A collection Π of permutations $\pi : \mathcal{U} \to \mathcal{U}$ is k-wise δ-dependent if for any distinct elements $x_1, \ldots, x_k \in \mathcal{U}$, the distribution on $(\pi(x_1), \ldots, \pi(x_k))$ induced by sampling $\pi \in \Pi$ is δ-close in statistical distance to the distribution induced by a truly random permutation. Arbitman *et al.* [ANS10] show how one can obtain succinct k-wise δ-dependent permutations that can be evaluated in constant time by combining the constructions in [NR99, KNR09]. Setting $k = n^{1/10} + n^{3/40}$ and $\delta = 1/n^{\Theta(1)}$ ensures that the bound on the size of the spare holds whp in each part and hence, by union bound, in all parts simultaneously.

To complete the proof, we need to prove that the partitioning is "balanced" whp also with respect to multisets. (Recall, that the cardinality of a multiset equals the sum of multiplicities of the elements in the support of the multiset.) Formally, we prove that the pseudo random partition induces in each part a multiset of cardinality at most $n^{1/10} + n^{3/40} \log^{3/2} n$ whp. As "heavy" elements of multiplicity at least $\log^3 n$ are stored in the spare, we may assume that multiplicities are less that $\log^3 n$.

We first describe how the partitioning is achieved in [ANS10]. The binary representation of x is partitioned into the leftmost $\log M$ bits, denoted by x_L and the remaining bits, denoted by x_R. A k'-wise independent hash function $f : \{0,1\}^{\log(u/M)} \to \{0,1\}^{\log M}$ is then sampled, with $k' = \lfloor n^{1/20}/(e^{1/3}) \rfloor$. The permutation π is defined as $\pi(x) = (x_L \oplus f(x_R), x_R)$.

Note that this induces a view of the universe as a two-dimensional table with u/M rows (corresponding to each x_R value) and M columns (corresponding to each $x_L \oplus f(x_R)$ value). Indeed, each cell of the table has at most one element (i.e., if $x = (x_L, x_R)$ and $y = (y_L, y_R)$ satisfy $x_L \oplus f(x_R) = y_L \oplus f(y_R)$ and $x_R = y_R$, then $x = y$). We define a *part* of the input multiset as consisting of all the elements of the input multiset that belong to the same column. The index of the part that x is assigned to is $x_L \oplus f(x_R)$. The corresponding part stores x_R.

The following observation follows from [ANS10, Claim 5.4] and the fact that the maximum multiplicity of each element is strictly less than $\log^3 n$.

Observation 9. *The cardinality of every part of the multiset is at most $n^{1/10} + n^{3/40} \log^{3/2} n$ whp.*

Proof. Fix a part $j \in [M]$ and for each $i \in [u/M]$, let $\mathcal{M}_i$ denote the multiset of all elements x with the x_R value equal to i (i.e., the multisets $\mathcal{M}_i$ consist of all the elements in row i). Each multiset $\mathcal{M}_i$ contributes at most one distinct element to the multiset of part j. Define $X_i \in [\log^3 n]$ to be the random variable that denotes the multiplicity of the element from $\mathcal{M}_i$ that is mapped to part j. Then $\mathbb{E}[X_i] = \frac{1}{M} \sum_{x \in \mathcal{U}} \mathcal{M}_i(x)$. Now define $X = \sum_{i \in [u/m]} X_i$ to be the random variable that denotes the cardinality of the multiset that is mapped into part j. By linearity of expectation, $\mathbb{E}[X] = n/M = n^{1/10}$. The random variables $\{X_i\}_i$ are k'-wise independent, since each variable X_i is determined by a different row in the table (and hence, each $\{X_i\}_i$ depends on a different x_R value). We scale the RVs $\{X_i\}_i$ by $\log^3 n$ and then apply Chernoff's bound for k'-wise independent

RVs [SSS95] and obtain:

$$\Pr\left[\frac{X}{\log^3 n} \geq \left(1 + \frac{\log^{3/2} n}{n^{1/40}}\right) \cdot \frac{n^{9/10}}{\log^3 n}\right] \leq \exp(-\lfloor k'/2 \rfloor) = \exp(-\Omega(n^{1/20})) \ .$$

The claim follows. □

6   The Counting Filter

To obtain a counting filter from our dictionary for multisets, use a pairwise independent hash function $h : \mathcal{U} \rightarrow [n/\varepsilon]$ to map an element x to a fingerprint $h(x)$ [CFG+78]. Let $\mathcal{M}_h$ denote the multiset over $[n/\varepsilon]$ induced by a multiset $\mathcal{M}$ over $\mathcal{U}$ defined by $\mathcal{M}_h(y) \triangleq \sum_{x:h(x)=y} \mathcal{M}(x)$. A multiset dictionary for $\mathcal{M}_h$ constitutes a counting filter in which the probability of an overcount is at most ε. The counting filter is requires $(1+o(1))\log(1/\varepsilon)n+O(n)$ bits and performs all operations in constant time. This completes the proof of Corollary 2.

References

[ANS09]   Yuriy Arbitman, Moni Naor, and Gil Segev. De-amortized cuckoo hashing: Provable worst-case performance and experimental results. In *International Colloquium on Automata, Languages, and Programming*, pages 107–118. Springer, 2009.

[ANS10]   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.

[BB08]   Daniel K Blandford and Guy E Blelloch. Compact dictionaries for variable-length keys and data with applications. *ACM Transactions on Algorithms (TALG)*, 4(2):1–25, 2008.

[BE20]   Ioana Oriana Bercea and Guy Even. A dynamic space-efficient filter with constant time operations. *CoRR*, abs/2005.01098, 2020. to appear in SWAT 2020.

[BM01]   Andrei Broder and Michael Mitzenmacher. Using multiple hash functions to improve ip lookups. In *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No. 01CH37213)*, volume 3, pages 1454–1463. IEEE, 2001.

[BMP+06] Flavio Bonomi, Michael Mitzenmacher, Rina Panigrahy, Sushil Singh, and George Varghese. An improved construction for counting Bloom filters. In *European Symposium on Algorithms*, pages 684–695. Springer, 2006.

[CFG+78] Larry Carter, Robert Floyd, John Gill, George Markowsky, and Mark Wegman. Exact and approximate membership testers. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 59–65. ACM, 1978.

[CM03]   Saar Cohen and Yossi Matias. Spectral Bloom filters. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 241–252, 2003.

[DadH90]   Martin Dietzfelbinger and Friedhelm Meyer auf der Heide. A new universal class of
             hash functions and dynamic hashing in real time. In *International Colloquium on
             Automata, Languages, and Programming*, pages 6–19. Springer, 1990.

[DadHPP06]   Erik D Demaine, Friedhelm Meyer auf der Heide, Rasmus Pagh, and Mihai Pătraşcu.
             De dictionariis dynamicis pauco spatio utentibus. In *Latin American Symposium on
             Theoretical Informatics*, pages 349–361. Springer, 2006.

[DDMM05]   Ketan Dalal, Luc Devroye, Ebrahim Malalla, and Erin McLeish. Two-way chaining
             with reassignment. *SIAM Journal on Computing*, 35(2):327–340, 2005.

[DR09]   Martin Dietzfelbinger and Michael Rink. Applications of a splitting trick. In *In-
             ternational Colloquium on Automata, Languages, and Programming*, pages 354–365.
             Springer, 2009.

[DW07]   Martin Dietzfelbinger and Christoph Weidling. Balanced allocation and dictionaries
             with tightly packed constant size bins. *Theoretical Computer Science*, 380(1-2):47–68,
             2007.

[Eli74]   Peter Elias. Efficient storage and retrieval by content and address of static files.
             Journal of the ACM (JACM), 21(2):246–260, 1974.

[FCAB00]   Li Fan, Pei Cao, Jussara Almeida, and Andrei Z Broder. Summary cache: a scal-
             able wide-area web cache sharing protocol. *IEEE/ACM transactions on networking*,
             8(3):281–293, 2000.

[FPSS05]   Dimitris Fotakis, Rasmus Pagh, Peter Sanders, and Paul Spirakis. Space efficient
             hash tables with worst case constant access time. *Theory of Computing Systems*,
             38(2):229–248, 2005.

[KM07]   Adam Kirsch and Michael Mitzenmacher. Using a queue to de-amortize cuckoo hash-
             ing in hardware. In *Proceedings of the Forty-Fifth Annual Allerton Conference on
             Communication, Control, and Computing*, volume 75, 2007.

[KNR09]   Eyal Kaplan, Moni Naor, and Omer Reingold. Derandomized constructions of k-wise
             (almost) independent permutations. *Algorithmica*, 55(1):113–133, 2009.

[Knu73]   Donald E Knuth. The art of computer programming, vol. 3: Searching and sorting.
             Reading MA: Addison-Wisley, 1973.

[LP10]   Shachar Lovett and Ely Porat. A lower bound for dynamic approximate membership
             data structures. In *2010 IEEE 51st Annual Symposium on Foundations of Computer
             Science*, pages 797–804. IEEE, 2010.

[NR99]   Moni Naor and Omer Reingold. On the construction of pseudorandom permutations:
             Lubyrackoff revisited. *Journal of Cryptology*, 12(1):29–66, 1999.

[Pag01]   Rasmus Pagh. Low redundancy in static dictionaries with constant query time. *SIAM
             Journal on Computing*, 31(2):353–363, 2001.

[Pan05]   Rina Panigrahy. Efficient hashing with lookups in two memory accesses. In *Proceed-
             ings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages
             830–839. Society for Industrial and Applied Mathematics, 2005.

[PPR05] Anna Pagh, Rasmus Pagh, and S. Srinivasa Rao. An optimal Bloom filter replacement. In *SODA*, pages 823–829. SIAM, 2005.

[PR01] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. In *European Symposium on Algorithms*, pages 121–133. Springer, 2001.

[PT14] Mihai Pătraşcu and Mikkel Thorup. Dynamic integer sets with optimal rank, select, and predecessor search. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 166–175. IEEE, 2014.

[RR03] Rajeev Raman and Satti Srinivasa Rao. Succinct dynamic dictionaries and trees. In *International Colloquium on Automata, Languages, and Programming*, pages 357–368. Springer, 2003.

[Sie04] Alan Siegel. On universal classes of extremely random constant-time hash functions. *SIAM Journal on Computing*, 33(3):505–543, 2004.

[SSS95] Jeanette P Schmidt, Alan Siegel, and Aravind Srinivasan. Chernoff–hoeffding bounds for applications with limited independence. *SIAM Journal on Discrete Mathematics*, 8(2):223–250, 1995.

A Implementation of the First Level of the Dictionary

In this section, we discuss two implementations of the first level of the dictionary that meet the specifications from Sec. 4.2. Namely, that of using global lookup tables like it was suggested in [ANS10] or an Elias-Fano encoding [Eli74]. We briefly review them here (for details, see [BE20]).

A.1 Global Lookup Tables

In this implementation, all bin and counter dictionaries employ common global lookup tables. Hence, it is sufficient to show that the size of the tables is $o(n)$. Each bin dictionary stores at most $n_B = (1 + \delta)B$ distinct elements from a universe $\mathcal{U}'$ of size $u' = u/(n/B)$. Therefore, the total number of states of a bin dictionary is $s \triangleq \binom{u'}{n_B}$. Each operation on the bin dictionary (query, insert, delete) is implemented as a function from $s \times u'$ to s., Namely, given the current state of the dictionary and an element $x \in \mathcal{U}'$, each function returns an updated state (in the case of insert(x) and delete(x)) or a bit (in the case of a membership query). The global lookup tables explicitly represent these functions and can be built in advance. Operations are therefore supported in constant time.

Moreover, each table requires at most $s \cdot u' \cdot \log s$ bits. Recall that we are in the sparse case defined as the case in which the size of the universe is small relative to n. Specifically, we have that $\log(u/n) = O(\log \log n)$, hence $u = n\,\mathsf{polylog}(n)$. Since $B = (\log n)/(\log(u/n))$ and $n_B = (1 + \delta)B$, one can show that, under these parametrizations, $s = \mathsf{polylog}(n) \cdot \sqrt{n}$ and the total number of bits each table takes is $o(n)$.

Similarly, we can build a lookup table that encodes the lexicographic order of the elements in each state of a BD. Each operation on the counter dictionaries is implemented by further indexing the lookup tables with an index that denotes the position of $c(x)$ in the CD.

A.2 Elias-Fano encoding

In this section, we briefly discuss the Elias-Fano encoding proposed in [BE20]. A bin dictionary implemented using this encoding is referred to as a "pocket dictionary". The idea is to represent each element in the universe $[u']$ as a pair (q, r), where $q \in [B]$ (the quotient) and $r \in [u'/B]$ (the remainder). A header encodes in unary the number of elements that have the same quotient. The body is the concatenation of remainders. The space required is $B + n_B(1 + \log(u/n))$ bits, which meets the required space bound since $B = O(n_B)$. Similarly, a counter dictionary can be implemented by storing the counters consecutively using an "end-of-string" symbol. We use a ternary alphabet for this encoding, which requires at most $\Theta(B)$ bits to encode each CD.

Both the BDs and the CDs fit in $O(1)$ words. Operations in the BD and the CD require rank and select instructions. See [BE20] for a discussion of how these operations can be executed in constant time if the RAM model can evaluate in constant time instructions represented as Boolean circuits with $O(\log w)$ depth and $O(w^2)$ gates.