Towards an Analysis of Quadratic Probing

William Kuszmaul
Harvard University, Cambridge, MA, USA

Zoe Xi
Massachusetts Institute of Technology, Cambridge, MA, USA

Abstract

Since 1968, one of the simplest open questions in the theory of hash tables has been to prove *anything* nontrivial about the correctness of quadratic probing. We make the first tangible progress towards this goal, showing that there exists a positive-constant load factor at which quadratic probing is a constant-expected-time hash table. Our analysis applies more generally to any fixed-offset open-addressing hash table, and extends to higher load factors in the case where the hash table examines blocks of some size $B = \omega(1)$.

2012 ACM Subject Classification Theory of computation → Sorting and searching

Keywords and phrases quadratic probing, hashing, open addressing, witness trees

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.103

Category Track A: Algorithms, Complexity and Games

Funding This work was partially sponsored by the United States Air Force Research Laboratory and the United States Air Force Artificial Intelligence Accelerator and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the United States Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.
William Kuszmaul: Funded by Harvard Rabin Postdoctural Fellowship.
Zoe Xi: Funded by the Carl W. Hoffman and Elizabeth B. Klerman Fund, and by the John Reed Fund.

1 Introduction

The field of open-addressed hash tables began with the introduction of linear probing in the 1950s [35, 24, 25]. Although early work [35] conjectured that linear probing should scale well to high load factors, with an insertion time of $O(x)$ at load factor $1-1/x$, subsequent analyses by Knuth [24] (unpublished) and by Konheim and Weiss [26] (published in 1966) showed that this is not the case. Due to *clustering effects*, in which elements group together to form long continuous runs of occupied slots, the true expected insertion time is asymptotically larger than researchers had hoped for, evaluating to $\Theta(x^2)$.

In the late 1960s, this prompted researchers to propose alternative hashing algorithms that preserved the simplicity (and in some cases data locality) of linear probing, while mitigating the clustering effects. Two solutions, in particular, emerged as natural alternatives, **double hashing**, which was introduced by de Balbine in his 1968 thesis [4] (and proposed independently by Bell and Kaman in 1970 [7]); and **quadratic probing**, which was introduced by Maurer in 1968 [32] and then refined by other sets of authors through the 1970s [5, 22, 16, 36].

Despite their data-structural simplicity, double hashing and quadratic probing proved far harder to analyze than linear probing. It wasn't until 1976, in a breakthrough paper by Guibas and Szemeredi [19], that double hashing was finally partially analyzed: they proved that, so long as the load factor of the hash table is at most ≈ 0.28, the hash table is

guaranteed to support constant-expected-time operations. This constant was subsequently improved to ≈ 0.31 in 1978 [20], which remained the state of the art until 1988, when Lueker [30, 31] finally extended the analysis to apply to all load factors. In particular, Lueker showed a coupling between double hashing and uniform probing, proving that the expected insertion times are within a $1 + o(1)$ factor of each other.

The coupling techniques [19, 20, 21, 30, 31] that allowed for an analysis of double hashing do not extend to quadratic probing. It has remained an open question for more than five decades to prove *anything* nontrivial about the behavior of quadratic-probing hash tables. It is not even known, for example, whether quadratic-probing is a constant-time data structure when used at a load factor of 0.001.

Linear Probing vs Double Hashing vs Quadratic Probing

Let us take a moment to briefly define the three hash-table designs described above. In each case, elements are stored in an array of some size n, and the **load factor** of the hash table is defined to be the fraction of slots that are occupied. To insert an element x, a sequence $p_1(x), p_2(x), \ldots$ of array positions are examined until an unoccupied spot is found for x. Where the three hash-table designs differ is in the choice of probe sequence: linear probing uses $p_i(x) = h(x) + i \mod n$, where $h(x) \in [n]$ is a random hash; double hashing uses $p_i(x) = h_1(x) + ih_2(x) \mod n$, where $h_1(x), h_2(x) \in [n]$ are both random hashes; and quadratic probing uses $p_i(x) = h(x) + (i-1)^2 \mod n$, where again $h(x) \in [n]$ is a random hash.

The analysis of linear probing [24, 26] hinges on the observation that, if an insertion x takes time k, there must be an array interval of the form $I = [h(x) - j, h(x) + k - 1]$, for some $j \geq 0$, such that the number of elements y with hashes $h(y) \in I$ is at least $|I| = j + k$. Thus, the analysis of linear probing reduces directly to the analysis of how many elements hash into each interval in the hash table. The analysis of double hashing [19, 20, 21, 30, 31] relies on the fact that the probe sequence $p_i(x) = h_1(x) + ih_2(x) \mod n$ is in some formal sense nearly as random as the fully random probe sequence $p_i(x) = h_i(x)$, where $h_1, h_2, \ldots$ are all independent hash functions.

Quadratic probing, on the other hand, sits in an unfortunate middle ground. It lacks the clean interval-based structure of linear probing – if an insertion x collides with another element y in position $p_i(x) = h(x) + (i-1)^2 \mod n$, then there is nothing substantive that we can say about the array interval $[h(x), h(y)]$. But it is not comparable to a fully random probe sequence – indeed, elements x and y with hashes $h(x)$ and $h(y)$ that are close together, are far more likely to interact than are a random pair of elements. The interactions between pairs of elements follow an apparently chaotic combinatorial structure: if an insertion x collides with another element y in position $p_i(x) = h(x) + (i-1)^2 \mod n$, then the same element x would not have interacted with y at all if y's hash had been 1 smaller; and, similarly, y might have been in a different position entirely had any of the elements *it* interacted with had even slightly different hashes, etc.

Yet, empirically, quadratic probing is an excellent hash table design [38, 32, 11, 15, 12, 6, 18, 23, 39, 40]. It preserves much of the data locality that makes linear probing special [38] while also empirically eliminating the asymptotic clustering effects that make linear probing bad [32]. It is recommended in textbooks and courses [11, 15, 12, 6, 18, 23, 39, 40], and is even used as the underlying design (with some modifications we will discuss later) for Google's in-house and open-source hash tables [1]. Thus, the problem would truly seem to be one of algorithm *analysis* rather than one of algorithm *design*.

This Paper: A Partial Analysis of Quadratic Probing

We give the first analysis for quadratic-probing hash tables at low load factors. We show that, at any load factor less than roughly 0.089, the expected time per operation is $O(1)$. In fact, our analysis applies not just to quadratic probing, but to any **fixed-offset** probing scheme, i.e., to any hash table that, like linear and quadratic probing, inserts elements via a probe sequence of the form $p_i(x) = h(x) + f(i)$ for some $f(i)$.

▶ **Theorem 1.** *There exists a positive constant $\alpha \geq 0.089$ such that all fixed-offset open addressing hash tables support constant-time insertions at load factor α and below. Moreover, the insertion time is bounded above by a geometric random variable with mean $O(1)$.*

The proof of Theorem 1 in Section 3 is achieved by a witness argument in which we construct two objects (a witness set S and a witness transcript T) that must exist in order for the insertion time to be large. The witness set S has the property that each individual option for S has only a very small probability of occurring. The witness transcript T, on the other hand, has the property there are only a relatively small number of options for what T can be. Finally, the relationship between the two (and, in particular, the fact T can be used to recover S), allows us to bound the probability of such a pair (S, T) existing at all.

The specific constant, 0.089, that we get from Theorem 1 stems from the careful enumeration of a family of strings that we call **witness strings** (i.e., candidates for the transcript T). Through a mixture of algebraic and combinatorial arguments, we obtain tight bounds on the growth rate for the family – this rate, in turn, dictates the best constant that we can get in our proof of Theorem 1.

Finally, in Section 4, we turn our attention to **chunked** fixed-offset open addressing, in which the probe sequence used is actually of the form

$$p_i(x) = h(x) + B \cdot f(\lfloor i/B \rfloor) + (i - 1)$$

for some **chunk size** B. The use of chunking is quite common in practice, as it reduces cache misses and allows for the use of hardware vectorization. A notable example is the hash table used at Google [1], which uses chunked quadratic probing with chunk size $B = 16$.

Our final result is an analysis of any chunk fixed-offset open-addressing scheme. We show that, when $B = \omega(1)$, such schemes can successfully handle load factors of the form $1 - o(1)$.

▶ **Theorem 2.** *There exists a constant $\alpha \in (0, 1)$ such that the following is true. Consider a chunked fixed-offset open-addressed hash table with chunk size B. Any insertion of an element x at a load factor $\alpha = 1 - 1/q$ satisfying $q \leq \alpha\sqrt{B/\log B}$ takes expected time at most $O(q^2)$. Moreover, the insertion time is bounded above by a geometric random variable with mean $O(q^2)$.*

We remark that, in general, the time/space tradeoff in Theorem 2 is nearly tight in the sense that linear probing (which is a trivial example of chunked open addressing) does, indeed, require $\Theta(q^2)$ time per insertion.

Other related work

In the half century since quadratic probing and double hashing were introduced, there has been a great deal of additional work on hash-table design. Notable examples include Cuckoo hashing [34], which allows for worst-case bounds on query time (and which has the interesting feature that, depending on the parameters with which it is implemented, there are genuine load-factor thresholds above which it cannot be used [34, 17, 14]); Robin-Hood

hashing [2, 10], which reorders elements in ways that reduce query time; graveyard hashing [9], which strategically leaves gaps within a linear probing hash table to reduce clustering; and many others [25]. In addition to these relatively practical designs, there has also been a great deal of progress on the theoretical extremes of how space/time efficient a hash table can be [29, 3, 13, 37]. In fact, recent works by Bender et al. [8] and by Li et al. [27, 28] close off the basic theoretical question of how space-efficient a hash table can be subject to a given set of time bounds. For a detailed but somewhat out-of-date survey, see Knuth's [25] 1998 edition of the Art of Computer Programming, Vol. 3.

2 Preliminaries

Let $h : U \to [n]$ be a random hash function, and let $r_1, r_2, \ldots, r_{n-1} \in \mathbb{N}$ be a permutation of the numbers 1 through $n-1$. Consider an open-addressing hash table with capacity n in which, to insert an element $x \in U$, we place it in the first available position from the sequence $h(x), h(x)+r_1, h(x)+r_2, \ldots$ (the positions wrap around, so position $n+1$ is the same as position 1). Such a hash table is said to perform **fixed-offset open-addressing** with **offset sequence** $r_1, r_2, \ldots$. The hash table is said to support **constant time insertions** at load factor $0 < \alpha \leq 1$ and below if, when the hash table is filled to a α-fraction full, each of the insertions is guaranteed to take constant expected time.

Additionally, a fixed-offset open-addressing scheme is said to be **chunked** (with **chunk size** B) if the $\{r_i\}$'s are broken into consecutive blocks of size B. That is, for each $i \geq 0$, and $j \in \{0, \ldots, B-1\}$, we have $r_{iB+j} = r_{iB} + j$.

One subtlety in the definition of fixed-offset open-addressing is the requirement that $r_1, r_2, \ldots, r_{n-1}$ form a *permutation* of $[n-1]$. We will see that, if our analysis is taking place at load factor $1 - 1/q$, the time per insertion is at most a geometric random variable with mean $O(1+q^2)$ (see Theorems 1 and 2), meaning that with probability $1 - 1/\operatorname{poly}(n)$, the hash table only ever uses the first $O((1+q^2)\log n)$ terms of any probe sequence. Therefore it is not strictly necessary to require that $r_1, r_2, \ldots, r_{n-1}$ are all distinct. It suffices for the probe sequence to satisfy the weaker requirement that $r_1, r_2, \ldots, r_{O((1+q^2)\log n)}$ are distinct, and to assume that the hash table is rebuilt from scratch if any operation ever takes $\omega((1+q^2)\log n)$ time. This distinction is important since the probe sequence used by quadratic probing is not a permutation for all table-sizes n [25], but is trivially guaranteed to have distinct entries for all of its first $\Omega(\sqrt{n})$ terms.

Finally, the reader may wonder whether our analyses can be extended to support deletions in addition to insertions and queries. Here there is a larger issue: quadratic probing does not natively *support* deletions. If one tries to implement deletions by simply removing items, then the query algorithm gets broken (it can no longer terminate when it sees a free slot) [25]. The standard way to implement deletions while preserving the correctness of queries is to use tombstones [9], which formally reduce the problem to the insertion-only setting.

3 Analysis for Sufficiently Small Constant Load Factors

In this section, we will show that there exists a universal load factor $\alpha \geq 0.089$ below which all fixed-offset open-addressing schemes are guaranteed to achieve $O(1)$-time operations.

▶ **Theorem 1.** *There exists a positive constant $\alpha \geq 0.089$ such that all fixed-offset open addressing hash tables support constant-time insertions at load factor α and below. Moreover, the insertion time is bounded above by a geometric random variable with mean $O(1)$.*

▶ **Corollary 3.** *There exists a positive constant $\alpha \geq 0.089$ such that quadratic probing supports constant-time insertions at load factor α and below.*

Let $r_1, r_2, \ldots$ be the offset sequence used by the hash table, and as a convention set $r_0 = 0$. Consider a sequence of up to αn insertions, followed by one additional insertion of an element x. Let D denote the state of the hash table when x is inserted, and for each element x that was inserted in the past, let $\text{index}(x, D)$ be the index i such that x resides in position $h(x) + r_i$.

Given a set $S \subseteq [n]$ we say that an element $x \in D$ **conflicts** with S if $h(x) \notin S$ and if $h(x) + r_i$ is in S for some $i \leq \text{index}(x, D)$. Moreover, if $h(x) + r_k$ is the first element of $h(x), h(x) + r_1, h(x) + r_2, \ldots$ to appear in the set S, then we say that the **pair** (x, k) **conflicts with** S **at position** $j = h(x) + r_k$. Finally, we define $\texttt{Conflicts}(S, j)$, which is referred to as a conflict set, to be the set of pairs that conflict with S at position $j \in S$, i.e.,

$$\texttt{Conflicts}(S, j) = \{(x, k) \mid x \in D, \\ 1 \leq k = \min\{i \mid h(x) + r_i \in S\},\ h(x) + r_k = j,\ k \leq \text{index}(x, D)\}.$$

We want to bound the probability that, when insertion x occurs, all of positions $h(x), h(x)+r_1, \ldots, h(x)+r_{\ell-1}$ are already occupied for some large ℓ, i.e., the insertion takes time greater than ℓ. Using the idea of a conflict set, we design a protocol BuildWitnesses (Algorithm 1) that takes as inputs D, $h(x)$, and ℓ, and returns a **witness set** S along with a **witness transcript** T. The witness set S will be a subset of $[n]$, and the witness transcript T will be a trinary string.

As we shall see, the basic idea is that, if the insertion of $h(x)$ into D takes time at least ℓ, then the $\text{BuildWitnesses}(D, h(x), \ell)$ protocol will return a pair (S, T) such that:

- the set $S \subseteq [n]$ is quite large, satisfying $|S| \geq \ell$;
- there are at least $|S|$ elements $x \in D$ satisfying $h(x) \in S$;
- the set S is fully determined by the triple $(h(x), \ell, T)$;
- and the transcript T is a trinary string of length $O(|S|)$.

We will then be able to argue that the probability of such a pair of objects existing is very small, at most $2^{-\Omega(\ell)}$. Thus, by analyzing BuildWitnesses, we will be able to indirectly arrive at a proof of Theorem 1. We emphasize that, although this approach uses an algorithm (BuildWitnesses) as part of the analysis, it is not an algorithm that actually gets executed by the hash table – it is simply for the sake of analysis.

Before we can dive into the analysis, we must show that Algorithm 1 terminates.

▶ **Lemma 4.** *Algorithm 1 terminates within finite time.*

Proof. First observe that Line 14 only adds elements to S that are not already in S. Since elements are never removed from S, it follows that each $j \in [n]$ can also be added to S (and therefore to `Unprocessed`) at most once. Since each phase (i.e., each iteration of Line 4) removes an item from `Unprocessed`, there can be at most n phases. Furthermore, since each iteration of Lines 11–15 increases the size of S, there can be at most n total iterations of lines 11–15. Therefore, the algorithm completes its construction of the witness objects S and T within $O(n)$ time. ◀

Next, we turn our attention to establishing the properties of S and T, along with their relationship to one another. Specifically, we will need the following three lemmas.

▶ **Lemma 5.** *Suppose the insertion x takes time greater than ℓ (that is, the length of the probe sequence is greater than ℓ). Then the witness set S has size at least ℓ, and at least $|S|$ elements $x \in D$ have hashes $h(x) \in S$.*

Algorithm 1

```
1:  procedure BuildWitnesses(D, h(x), ℓ)
2:      Set Unprocessed = {h(x), h(x) + r_1, h(x) + r_2, ..., h(x) + r_{ℓ−1}}.
3:      Set S = Unprocessed and T = (0)^{ℓ−1} ∘ 1.
4:      while |Unprocessed| > 0 do                    ▷ Each iteration is called a phase.
5:          Let j = max{Unprocessed}.
6:          Remove j from Unprocessed.
7:          Append 2 to T.
8:          while true do
9:              Let Conflicts(S, j) = {(x, k) | x ∈ D, 1 ≤ k = min{i | h(x) + r_i ∈ S},
10:                                                  h(x) + r_k = j, k ≤ index(x, D)}.
11:             if |Conflicts(S, j)| > 0 then
12:                 Let k = max_k{(x, k) ∈ Conflicts(S, j) for some x}.
13:                 Let x be an arbitrary element in {x | (x, k) ∈ Conflicts(S, j)}.
14:                 Add h(x), h(x) + r_1, h(x) + r_2, ..., h(x) + r_{k−1} to Unprocessed and to S.
15:                 Append (0)^{k−1} ∘ 1 to T.
16:             else
17:                 End while loop.
18:     return (S, T)
```

Proof. The fact that S has size at least ℓ is immediate from the initialization of S in BuildWitnesses. To prove that at least $|S|$ elements $x \in D$ have hashes $h(x) \in S$, we will show a stronger claim: that every position in S is occupied by an element x whose hash $h(x)$ is in S.

First, observe that, by design, every position in S is occupied. Indeed, by the assumption that x's insertion takes time greater than ℓ, we know that $h(x), h(x) + r_1, h(x) + r_2, \ldots, h(x) + r_{\ell-1}$ (the positions initially placed in S) are all occupied. And by the definition of $\texttt{Conflicts}(S, j)$, we know that the positions added by Line 14 are also always occupied.

Now suppose for contradiction that some position $s \in S$ contains an element x whose hash $h(x)$ is not in S. Then x must conflict with S, and, in particular, x must be part of a pair (x, k) that conflicts with S at some position $j \in S$. Now consider the phase that processed j, and define S' to be the state of S at the end of the phase. Because the phase ended, we must have had $\texttt{Conflicts}(S', j) = \emptyset$. But, because (x, k) conflicts with S at position j, and since $j \in S' \subseteq S$, it must be that (x, k) also conflicts with S' at position j. This means that $(x, k) \in \texttt{Conflicts}(S', j)$, which is a contradiction. ◀

▶ **Lemma 6.** *Given the witness transcript T, along with $h(x)$ and ℓ, one can reconstruct the witness set S.*

Proof. This is accomplished by the WitnessStringToSet protocol (Algorithm 2). The basic idea is that we can use the witness transcript T to simulate the execution of BuildWitnesses. Namely, we can use 2s in T to determine boundaries between phases; and then we can use runs of 0s in T to determine the value of k that is used in each iteration of the inner while loop. This allows for us to fully reconstruct the witness set S using just $T, h(x), \ell$. ◀

▶ **Lemma 7.** *The witness transcript T is a trinary string of length $2|S|$.*

Proof. Each time that we append a 2 to T, we remove an element from `Unprocessed`. We do this $|S|$ times, so there are $|S|$ 2s in T. Each time we append a string $(0)^{k-1} \circ 1$ to T, we also add k elements to S (and vice-versa). Thus, the total number of 0s and 1s in T is $|S|$. It follows that T is a trinary string of length $2|S|$. ◀

Algorithm 2

```
1: procedure WitnessStringToSet(h(x), ℓ, T)
2:     Set Unprocessed = {h(x), h(x) + r_1, h(x) + r_2, ..., h(x) + r_{ℓ−1}}.
3:     Set S = Unprocessed.
4:     Remove prefix (0)^{ℓ−1} ∘ 1 from T.
5:     while |Unprocessed| > 0 do
6:         Let j = max_{j∈Unprocessed} j.
7:         Remove j from Unprocessed.
8:         while first character of T is not a 2 do
9:             Let k − 1 be the number of 0s at the start of T before the first 1.
10:            Define g = j − r_k.
11:            Add g, g + r_1, g + r_2, ..., g + r_{k−1} to Unprocessed and to S.
12:            Remove first k characters of T, which are (0)^{k−1} ∘ 1.
13:        Remove first character of T, which is a 2.
14:    return S
```

In addition to these structural lemmas, we will need a basic concentration bound on the probability that $|S|$ elements hash to a set S of some size.

▶ **Lemma 8.** *Let $\alpha = 1/e - \Omega(1)$. Consider a set A of αn elements, each with random hashes in $[n]$. Let $B \subseteq [n]$ be a set of some size k. The probability that at least k elements from A have hashes in B is at most $\left((1+o(1))\alpha e^{(1-\alpha)}\right)^k$, where the o-notation is in terms of k.*

Proof. By a standard Poisson approximation (see, e.g., Theorem 5.10 in [33]), and because the event of at least k elements hashing to B is monotone (adding more elements never undoes the event), we have that the probability of at least k elements hashing to B is at most $2\Pr[\text{Poisson}(\alpha k) \geq k]$. This, in turn is

$$
\begin{aligned}
&2\sum_{j\geq k} \Pr[\text{Poisson}(\alpha k) = j] \\
&= 2\sum_{j\geq k} \frac{(\alpha k)^j e^{-\alpha k}}{j!} \\
&\leq 2\sum_{j\geq k} 2^{o(j)} (\alpha k)^j e^{-\alpha k}/(j^j/e^j) && \text{(by Stirling's approximation)} \\
&\leq 2\sum_{j\geq k} 2^{o(j)} \alpha^j e^{-\alpha k + j} \\
&= 2e^{-\alpha k}\sum_{j\geq k} (\alpha e \cdot (1+o(1)))^j \\
&= O\left(e^{-\alpha k}(\alpha e \cdot (1+o(1)))^k\right) && \text{(since } \alpha = 1/e - \Omega(1)\text{)} \\
&= \left((1+o(1))\alpha e^{1-\alpha}\right)^k. && \blacktriangleleft
\end{aligned}
$$

Putting these lemmas together, we can now prove a weak version of Theorem 1 in which we do not seek to optimize the constant α.

▶ **Theorem 9.** *There exists a positive constant α such that all fixed-offset open addressing hash tables support constant-time insertions at load factor α and below. Moreover, the insertion time is bounded above by a geometric random variable with mean $O(1)$.*

Proof. Let us bound the probability that the insertion of x, which takes place at a load factor of at most α, takes time greater than $\ell > 0$. Let S and T be the witness set and witness transcript produced by BuildWitnesses$(D, h(x), \ell)$.

Suppose the insertion takes time greater than ℓ. Then, by Lemma 5, witness set S has some size $q \geq \ell$ and has the property that at least q elements $x \in D$ satisfy $h(x) \in S$.

Define $\mathcal{S}_q$ to be the set of options for what S could be if its size is q, conditioned on $h(x)$ and ℓ. For each $R \in \mathcal{S}_q$, let X_R be the event that $|\{x \in D \mid h(x) \in R\}| \geq q$. Then, in order for the insertion of x to take time greater than ℓ, event X_R must occur for some $R \in \bigcup_{q \geq \ell} \mathcal{S}_q$. By a union bound, the probability of this happening is at most

$$\sum_{q \geq \ell} \sum_{R \in \mathcal{S}_q} \Pr[X_R].$$

By Lemma 8, and assuming that $\alpha \leq e^{-1} - \Omega(1)$, this is at most

$$\sum_{q \geq \ell} \left(|\mathcal{S}_q| \cdot ((1 + o(1)) \cdot \alpha e^{1-\alpha})^q \right). \tag{1}$$

To bound $|\mathcal{S}_q|$, observe that by Lemma 6, each set $R \in \mathcal{S}_q$ corresponds to a unique witness transcript $T \in [3]^{2q}$. Thus $|\mathcal{S}_q| \leq 9^q$, which allows us to bound (1) by

$$\sum_{q \geq \ell} \left(9 \cdot (1 + o(1)) \cdot \alpha e^{1-\alpha} \right)^q. \tag{2}$$

Supposing that $\alpha e^{1-\alpha} < 1/9$, this sum evaluates to $e^{-\Omega(\ell)}$. In other words, the probability of the insertion x taking time greater than ℓ is exponentially small in ℓ. This means that the insertion takes $O(1)$ expected time. ◀

To improve upon the constant α and obtain the full version of Theorem 1, we will need a tighter bound on the number of witness transcripts corresponding to a witness set of size ℓ.

We begin by defining an infinite family of strings that contains all possible witness transcripts. Define a **witness phrase** to be a string of the form

$$P = (0)^{i_1} \circ 1 \circ (0)^{i_2} \circ 1 \circ \cdots \circ (0)^{i_k} \circ 1$$

where $0 \leq i_1 < i_2 < \cdots < i_k$. Define a **witness string** to be a string of the form

$$W = P_1 \circ 2 \circ P_2 \circ 2 \circ P_3 \circ \cdots \circ P_j \circ 2,$$

where each of $P_1, P_2, \ldots, P_j$ are witness phrases. The number j is referred to as W's **phrase count**, and the number of 0s and 1s in W is referred to as W's **zero/one count**.

Let $\mathcal{W}_{a,b}$ be the set of witness strings with zero/one-count a and phrase-count b. Notice, in particular, that $\mathcal{W}_{m,m}$ contains all witness transcripts that correspond to witness sets of size m. In Section 3.1, we will prove the following proposition, which characterizes the exact growth rate of $|\mathcal{W}_{m,m}|$.

Let $u \in (0, 1)$ minimize the quantity

$$f(u) = u^{-1} \prod_{i=1}^{\infty} (1 + u^i).$$

Then,

$$|\mathcal{W}_{m,m}| = f(u)^{(1-o(1))m}.$$

Note that, since $\log f(u) = \log u^{-1} + \sum_i \log(1+u^i)$ is within a constant factor of $\log u^{-1} + \sum_i u^i$, we know that $f(u)$ converges for any $u \in (0,1)$, and that $f(u)$ has a minimum value (since $f(u) \to \infty$ as either $u \to 0$ or $u \to 1$).

A manual calculation shows that $f(u)$ minimizes to slightly smaller than 4.51. It follows that the number of witness strings corresponding to witness sets of size m is at most $O(4.51^m)$. Plugging this into the proof of Theorem 9, (2) becomes

$$\sum_{q \geq \ell} \left(4.51 \cdot (1+o(1)) \cdot \alpha e^{1-\alpha}\right)^q.$$

In order so that, as before, this sum comes out to $e^{-\Omega(\ell)}$, we need that $\alpha e^{1-\alpha} < 1/4.51$. Another manual calculation tells us that it suffices to have $\alpha \leq 0.089$. With this modification to the proof of Theorem 9, we get Theorem 1.

3.1 Proof of Proposition 3

In this subsection, we complete the final component needed to prove Theorem 1, namely, the proof of Proposition 3, restated below.

Let $u \in (0,1)$ minimize the quantity

$$f(u) = u^{-1} \prod_{i=1}^{\infty} (1+u^i).$$

Then,

$$|\mathcal{W}_{m,m}| = f(u)^{(1-o(1))m}.$$

We begin by reinterpreting $|W_{k,m}|$ as the coefficient of x^k in a certain polynomial $G(x)$.

▶ **Lemma 10.** *Consider the formal power series*

$$G(x) = \left(\prod_{i=1}^{\infty}(1+x^i)\right)^m = \sum_i g_i x^i, \tag{3}$$

where the g_i's denote the coefficients of G, and where x is a formal variable. Then, the coefficient g_k is precisely equal to $|\mathcal{W}_{k,m}|$.

Proof. The coefficient of x^k in

$$\prod_{i=1}^{\infty}(1+x^i)$$

is equal to the number of witness phrases with zero/one-count k; and so the coefficient g_k of x^k in

$$\left(\prod_{i=1}^{\infty}(1+x^i)\right)^m$$

is equal to the number of ways to pick m witness phrases with total one-count k. This, in turn, is precisely $|\mathcal{W}_{k,m}|$. ◀

Using a simple argument about what happens when we evaluate the polynomial $G(x)$ at a given value $u \geq 0$, we can obtain the upper-bound side of Proposition 3.

▶ **Lemma 11.** *For any $u \geq 0$, we have*

$$|\mathcal{W}_{m,m}| \leq f(u)^m.$$

Proof. Let G be the formal power series from Lemma 10, and let $f(u)$ be the function defined in Proposition 3. For any $u \geq 0$, we have

$$f(u)^m = G(u)/u^m = \sum_i g_i u^i/u^m \geq g_m u^m/u^m = g_m = |\mathcal{W}_{m,m}|.$$ ◀

The lower bound is a bit more tricky. The main step is to re-interpret $G(px)/G(p)$ (where $p \in (0,1)$ and x is a formal variable) as the generating function for a random variable $X^{(p)}$ (this reinterpretation is a trick of the analysis, since $X^{(p)}$ does not appear anywhere in the original problem formulation), and then prove a concentration bound on that random variable.

We begin by establishing some basic conventions for discussing the generating function of a random variable. Supposing that A is a random variable that takes non-negative integer values, we define the **generating function** for A to be the formal power series

$$f(x) = \sum_{i \geq 0} \Pr[A = i] \cdot x^i.$$

We will make extensive use of two standard facts:

- **Fact 1:** Given any formal power series $f(x)$ such that $f(1)$ exists, the polynomial $f(x)/f(1)$ is the generating function for some random variable.
- **Fact 2:** Given two generating functions $f(x)$ and $g(x)$ for random variables A and B, the polynomial $f(x)g(x)$ is the generating function for the random variable C obtained by summing independent copies of A and B.

With these properties in mind, we now argue that, for the formal power series $G(x)$ from Lemma 11, there exists some $p \in \Theta(1) \cap (1 - \Theta(1))$ such that $G(px)/G(p)$ is the generating function for some well-behaved random variable. We will then be able to obtain our lower bound by analyzing the properties of this random variable.

▶ **Lemma 12.** *Consider the formal power series*

$$G(x) = \left(\prod_{i=1}^{\infty}(1 + x^i)\right)^m = \sum_i g_i x^i.$$

Then there exists $p \in \Theta(1) \cap (1 - \Theta(1))$ such that

$$G^{(p)}(x) = \frac{G(px)}{G(p)} = \sum_i g_i^{(p)} x^i$$

is the generating function for a random variable X with mean m and standard deviation $O(\sqrt{m})$.

Proof. By design, for any $p \in (0,1)$, we have that $\sum_i g_i^{(p)} = G^{(p)}(1) = 1$, so by Fact 1, $G^{(p)}(x)$ is the generating function for *some* random variable $X^{(p)}$. We will show that there exists some $p \in (0,1)$, satisfying $p = \Theta(1) \cap (1 - \Theta(1))$, such that $\mathbb{E}[X^{(p)}] = m$; and that for any $p = 1 - \Theta(1)$, the standard deviation of $X^{(p)}$ is $\Theta(\sqrt{m})$.

First observe that

$$G^{(p)}(x) = \frac{G(px)}{G(p)} = \left(\frac{\prod_{i=1}^{\infty}(1+(px)^i)}{\prod_{i=1}^{\infty}(1+p^i)}\right)^m.$$

It follows by Facts 1 and 2 that $X^{(p)}$ can be interpreted as the sum of m independent random variables $X_1^{(p)}, \ldots, X_m^{(p)}$ where each $X_j^{(i)}$ has generating function

$$\frac{\prod_{i=1}^{\infty}(1+(px)^i)}{\prod_{i=1}^{\infty}(1+p^i)}.$$

Observe that, when $p = 0$, we have $\mathbb{E}[X_i^{(p)}] = 0$, and as $p \to 1$, we have $\mathbb{E}[X_i^{(p)}] \to \infty$. Thus, there must exist some $p \in (0,1)$ such that $\mathbb{E}[X_i^{(p)}] = 1$. Since this choice of p is oblivious to m, it must be that $p = \Theta(1) \cap (1 - \Theta(1))$. This value of p gives us $\mathbb{E}[X^{(p)}] = m$, as desired.

Having chosen p, it remains to show that the standard deviation of $X^{(p)}$ is $O(\sqrt{m})$, or equivalently, that the variance of $X^{(p)}$ is $O(m)$. Since $X^{(p)} = \sum_{j=1}^{m} X_j^{(p)}$ is a sum of independent random variables, it suffices to show that each $X_j^{(p)}$ has variance $O(1)$. Notice, however, that the generating function for $X_j^{(p)}$ can be written as

$$\prod_{i=1}^{\infty} \frac{(1+(px)^i)}{(1+p^i)} = \prod_{i=1}^{\infty} \frac{H^{(p,i)}(x)}{H^{(p,i)}(1)},$$

where $H^{(p,i)}(x)$ is the formal power series $1 + (px)^i$. By Facts 1 and 2, it follows that we can interpret $X_j^{(p)}$ as a sum of independent random variables $Y_{j,1}^{(p)}, Y_{j,2}^{(p)}, \ldots$ where each $Y_{j,i}^{(p)}$ has generating function $\frac{(1+(px)^i)}{(1+p^i)}$. This means that

$$\mathrm{var}(X_j^{(p)}) = \sum_i \mathrm{var}(Y_{j,i}^{(p)}) \leq \sum_i \mathbb{E}[(Y_{j,i}^{(p)})^2] = \sum_i \frac{p^i i^2}{(1+p^i)} = O(1),$$

where the final equality uses $p = 1 - \Theta(1)$. As noted above, this implies that $X^{(p)}$ has variance $O(m)$, which completes the proof. ◀

We will also need a straightforward lemma bounding $|\mathcal{W}_{m\pm k,m}|$ in terms of $|\mathcal{W}_{m,m}|$ (multiplicatively) for small k.

▶ **Lemma 13.** *For any $0 \leq k < m$, we have that*

$$|\mathcal{W}_{m-k,m}| \leq |\mathcal{W}_{m,m}|$$

and that

$$|\mathcal{W}_{m+k,m}| \leq 2^{O(k \log m)} |\mathcal{W}_{m,m}|.$$

Proof. Given a string $s \in \mathcal{W}_{m-k,m}$, we can obtain a string $\phi(s) \in \mathcal{W}_{m,m}$ by taking the final run of 1s in s and extending the length of that run by k. This function is injective, implying that $|\mathcal{W}_{m-k,m}| \leq |\mathcal{W}_{m,m}|$.

Given a string $s \in \mathcal{W}_{m+k,m}$, it is possible to remove some set of $O(k)$ characters (zeros and ones) in order to obtain a valid string $s' \in \mathcal{W}_{m,m}$; let $\psi(s)$ be the lexicographically smallest such string s' that one can achieve in this way. For a given $s' \in \mathcal{W}_{m,m}$, we can bound $|\psi^{-1}(s')| \leq 2^{O(k \log m)}$, since there are at most $2^{O(k \log m)}$ ways to *add* $O(k)$ characters to get a string $s \in \mathcal{W}_{m+k,m}$. It follows that $|\mathcal{W}_{m+k,m}| \leq 2^{O(k \log m)} |\mathcal{W}_{m,m}|$. ◀

Finally, we can put the pieces together in order to get the lower-bound side of Proposition 3.

▶ **Lemma 14.** *Let*

$$f(u) = u^{-1} \prod_{i=1}^{\infty} (1 + u^i).$$

Then, there exists $p \in (0, 1)$ *such that*

$$f(p)^m \leq 2^{o(m)} |\mathcal{W}_{m,m}|.$$

Proof. Consider the formal power series

$$G(x) = \left(\prod_{i=1}^{\infty} (1 + x^i) \right)^m = \sum_i g_i x^i.$$

By Lemma 12, there exists $p \in \Theta(1) \cap (1 - \Theta(1))$ such that

$$G^{(p)}(x) = \frac{G(px)}{G(p)} = \sum_i g_i^{(p)} x^i$$

is the generating function for a random variable X with mean m and standard deviation $O(\sqrt{m})$. By Chebyshev's inequality, at least half of X's probability mass must be concentrated between $m - O(\sqrt{m})$ and $m + O(\sqrt{m})$. That is, for some positive constant d,

$$\sum_{k=-d\sqrt{m}}^{d\sqrt{m}} g_{m+k}^{(p)} \geq \frac{1}{2} \sum_{i=0}^{\infty} g_i^{(p)}.$$

Since $g_i^{(p)} = p^i g_i / G(p)$, we can multiply both sides by $G(p)$ to get

$$\sum_{k=-d\sqrt{m}}^{d\sqrt{m}} p^{m+k} g_{m+k} \geq \frac{1}{2} \sum_{i=0}^{\infty} p^i g_i.$$

Since $p = \Theta(1)$ and since $g_{m+k} = |\mathcal{W}_{m+k,m}| \leq 2^{O(|k| \log m)} |\mathcal{W}_{m,m}|$ (by Lemmas 10 and 13), we have that each term $p^{m+k} g_{m+k}$ in the left sum is at most

$$2^{O(\sqrt{m} \log m)} p^{m+k} |\mathcal{W}_{m,m}| = 2^{O(\sqrt{m} \log m)} p^m |\mathcal{W}_{m,m}|.$$

This means that the entire left sum is at most

$$O(\sqrt{m}) 2^{O(\sqrt{m} \log m)} |\mathcal{W}_{m,m}| \leq 2^{o(m)} p^m |\mathcal{W}_{m,m}|.$$

Therefore,

$$\sum_{i=0}^{\infty} p^i g_i \leq 2^{o(m)} p^m |\mathcal{W}_{m,m}|,$$

which implies that

$$G(p)/p^m \leq 2^{o(m)} |\mathcal{W}_{m,m}|.$$

Finally, since $G(p)/p^m = f(p)^m$, the proof is complete. ◀

4 Chunked Fixed-Offset Open Addressing

We now turn our attention to obtaining bounds for chunked fixed-offset open-addressed hash tables. Recall that a fixed-offset open-addressed hash table is said to be chunked with chunk-size B if, for each $i \geq 0$ and $j \in \{0, \ldots, B-1\}$, we have $r_{iB+j} = r_{iB} + j$. As terminology, for a given element x, we will refer to the sequence $h(x) + r_{iB}, \ldots, h(x) + r_{iB+B-1}$ as the ***i*-th block** in x's probe sequence. Our goal will be to prove the following theorem.

▶ **Theorem 2.** *There exists a constant $\alpha \in (0,1)$ such that the following is true. Consider a chunked fixed-offset open-addressed hash table with chunk size B. Any insertion of an element x at a load factor $\alpha = 1 - 1/q$ satisfying $q \leq \alpha\sqrt{B/\log B}$ takes expected time at most $O(q^2)$. Moreover, the insertion time is bounded above by a geometric random variable with mean $O(q^2)$.*

The theorem can be interpreted as saying that, whenever $B = \omega(1)$, it is possible to support load factors of the form $\alpha = 1 - q$ were $q = o(1)$. We remark that the time bound $O(q^2)$ is optimal in general, since linear probing is an example of a chunked open-addressed hash table that has insertion time $\Theta(q^2)$ at load factor $1 - q$.

We will prove Theorem 2 with a slightly more intricate analysis of the BuildWitnesses procedure from Section 3. To do this, we must first define the notion of an *analytical run*.

In lines 2 and 14 of BuildWitnesses, when we add some sequence $Q = h(x), h(x)+r_1, \ldots$ to `Unprocessed`, we can break the positions that we add into **analytical runs**, where the first (up to) B entries of Q form the first analytical run, the next (up to) B entries form the next analytical run, and so on. The execution of the line creates $\lceil |Q|/B \rceil$ analytical runs, and all but the final one have size B.

Say that two analytical runs r_1, r_2 (not necessarily created by the same iteration of Line 14) are **adjacent** if they represent sub-intervals of the form $[i, j]$ and $[j+1, k]$ for some i, j, k. Finally, define the **adjusted size** of an analytical run r to be B if the sequence Q that added r to S satisfied $|Q| \geq B$ or if the analytical run r was created by Line 2 of the algorithm, and define the adjusted size to be $|r|$ otherwise.

A critical step in the proof of Theorem 2 is to observe that, whenever two analytical runs are adjacent, their adjusted sizes must sum to at least B.

▶ **Lemma 15.** *Consider the execution of BuildWitnesses$(D, h(u), \ell)$ for some $\ell > 0$. Consider two adjacent analytical runs r_1, r_2 with adjusted sizes s_1, s_2. Then $s_1 + s_2 \geq B$.*

Proof. Let x_1 and x_2 be the elements that created runs r_1 and r_2. Let t_1 and t_2 be the iterations at which x_1 and x_2 are added to S by BuildWitnesses, let $\texttt{Conflicts}_1$ and $\texttt{Conflicts}_2$ be the conflict sets that are constructed in the while-loop iterations that add x_1 and x_2 to S, and let S_1 and S_2 be the states of S immediately before x_1 and x_2 are added to S, respectively.

If either of r_1 or r_2 have adjusted size B, then the claim is trivial. We can therefore assume without loss of generality that r_1, r_2 are the first runs in the probe sequences for x_1 and x_2 and that $|r_1|, |r_2| < B$. Note that, as immediate consequences, we have that r_1 and r_2 were created by Line 14 of BuildWitnesses (rather than Line 2); that $h(x_1) < h(x_2)$; and that x_2 was added to S before x_1 (i.e., $t_2 < t_1$).

Let ℓ be the right endpoint of r_2. We claim that

$$\ell \geq h(x_1) + B - 1. \tag{4}$$

This would imply that $|r_1| + |r_2| \geq \ell - h(x_1) + 1 \geq B$, completing the proof.

Suppose for contradiction that (4) does not hold. Let $(x_2, k_2) \in \texttt{Conflicts}_2$ be the pair containing x_2 in $\texttt{Conflicts}_2$. Then the final position ℓ of run r_2 satisfies $\ell + 1 = h(x_2) + k_2$, and $\texttt{Conflicts}_2 = \texttt{Conflicts}(S_2, \ell + 1)$ expands to

$$\{(x,k) \mid x \in D, \\ 1 \le k = \min\{i \mid h(x) + r_i \in S_2\},\ h(x) + r_k = \ell + 1,\ k \le \text{index}(x, D)\}.$$

We will show that this forces the pair $(x_1, \ell - h(x_1) + 1)$ to be in $\texttt{Conflicts}_2$. But the fact that $h(x_1) < h(x_2)$ means that BuildWitnesses will rather select $(x_1, \ell - h(x_1) + 1)$ than $(x_2, \ell - h(x_2) + 1)$ from $\texttt{Conflicts}_2$ (see Line 12 of BuildWitnesses). This contradicts the fact that x_2 is added to S at iteration t_2 in the execution of BuildWitnesses.

It remains to prove that $(x_1, \ell - h(x_1) + 1) \in \texttt{Conflicts}_2$. Since $\ell < h(x_1) + B - 1$ (by assumption) and since $h(x_1) < h(x_2)$, we know that $\ell + 1 = h(x_1) + k_1$ for some $k_2 < k_1 < B$. To prove that $(x_1, k_1) \in \texttt{Conflicts}_2$, we must show that

$$k_1 = \min\{i \mid h(x_1) + r_i \in S_2\} \tag{5}$$

and that

$$k_1 \le \text{index}(x_1, D). \tag{6}$$

We begin by showing (5). The fact that $(x_2, k_2) \in \texttt{Conflicts}_2$ tells us that $k_2 = \min\{i \mid h(x_2) + r_i \in S_2\}$, which implies that $h(x_2), h(x_2) + 1, \ldots, \ell \notin S_2$. The fact that $(x_1, h(x_2) - h(x_1)) \in \texttt{Conflicts}_1$ tells us that $h(x_2) - h(x_1) = \min\{i \mid h(x_1) + r_i \in S_1\}$, which implies that $h(x_1), h(x_1) + 1, \ldots, h(x_2) - 1 \notin S_1$. Since BuildWitnesses always processes $\max\{\texttt{Unprocessed}\}$, we know that, between iterations t_2 and t_1, BuildWitnesses processes only values of j satisfying $j \ge h(x_2)$. Thus $S_2 \cap [h(x_2) - 1] \subseteq S_1 \cap [h(x_2) - 1]$. So the fact that $h(x_1), h(x_1) + 1, \ldots, h(x_2) - 1 \notin S_1$ implies that $h(x_1), h(x_1) + 1, \ldots, h(x_2) - 1 \notin S_2$. Therefore, $h(x_1), h(x_1) + 1, \ldots, \ell \notin S_2$, which implies (5).

Finally, we complete the proof by establishing (6). If we had $k_1 = \ell + 1 - h(x_1) > \text{index}(x_1, D)$, then x_1 would have to reside in one of positions $h(x_1), h(x_1) + 1, \ldots, \ell$. If x_1 resides in any of positions $h(x_2), \ldots, \ell - 1$, then at time $t_2 < t_1$, that position must have been vacant, which implies that x_2 could have used it – but this contradicts the fact that x_2 created run r_2. On the other hand, if x_1 resides in any of positions $h(x_1), h(x_1) + 1, \ldots, h(x_2) - 1$, then x_1 does not conflict with position $h(x_2)$, which contradicts the fact that x_1 created run r_1. Thus, x_1 does not reside in any of positions $h(x_1), h(x_1) + 1, \ldots, \ell$, which means that $k_1 = \ell + 1 - h(x_1) \le \text{index}(x_1, D)$. ◀

We can build on Lemma 15 to make a claim about the average size of all of the analytical runs, namely that, if the analytical runs are not *all* adjacent, then their average (non-adjusted!) size must be $\Omega(B)$. It may see odd at first glance that we need to separate out the case where all of the analytical runs are adjacent, but we will see later on that this case actually behaves very differently from the other cases. (Indeed, it is the case that causes the insertion time to be $O(q^2)$ instead of $O(1)$!)

▶ **Lemma 16.** *Consider the execution of* BuildWitnesses$(D, h(u), \ell)$ *for some* $\ell > 0$. *Let* $r_1, r_2, \ldots, r_j$ *be the analytical runs that we add to* `Unprocessed` *during the execution of the algorithm. If* $r_1, r_2, \ldots, r_j$ *are not all adjacent, then* $\sum_{i=1}^{j} |r_i| \ge B$ *and*

$$\frac{1}{j} \sum_{i=1}^{j} |r_i| \ge \Omega(B).$$

Proof. Define $s_1, s_2, \ldots, s_j$ be the sizes of the analytical runs and let $s'_1, s'_2, \ldots, s'_j$ to be the adjusted sizes of the analytical runs. Since $r_1, r_2, \ldots, r_j$ are not all adjacent, we know that at least one r_i satisfies $|s_i| = B$ (namely, the first time we add an r_i that is not adjacent with the other r_js added so far). Therefore, the lemma is immediately true if $j = O(1)$. Suppose for the rest of the proof that $j = \omega(1)$.

With the possible exception of the first analytical run added by Line 2 (recall analytical runs created by Line 2 automatically have adjusted size B), we have that for every analytical run r whose true size s_i is smaller than its adjusted size s'_i, there is some run with true size B. Indeed, such a run r must be the final analytical run added by some iteration of Line 14, and that that iteration of Line 14 must have added at least one analytical run with true size B. It follows that

$$B + \sum_{i=1}^{j} s_i \geq \frac{1}{2} \sum_{i=1}^{j} s'_i.$$

Because $j = \omega(1)$, this implies

$$o(B) + \frac{1}{j} \sum_{i=1}^{j} s_i \geq \Omega\left(\frac{1}{j} \sum_{i=1}^{j} s'_i\right).$$

Thus, to complete the proof, it suffices to show that the right side is $\Omega(B)$.

Now consider a maximal sequence of adjacent analytical runs $r_{a_1}, r_{a_2}, r_{a_3}, \ldots, r_{a_k}$. If $k = 1$, then the adjusted size of the run r_{a_1} is guaranteed to be B (because either its true size is B and it was created by Line 14, or it was created by Line 2 which only creates analytical runs with adjusted sizes of B). If $k > 1$, then Lemma 15 tells us that the runs $r_{a_1}, r_{a_2}, r_{a_3}, \ldots, r_{a_k}$ have adjusted sizes summing to at least $\lfloor k/2 \rfloor \cdot B \geq kB/3$.

Therefore, the average adjusted size over all analytical runs is at least $B/3$. This means that

$$\frac{1}{2j} \sum_{i=1}^{j} s'_i \geq B/6,$$

which completes the proof. ◀

Finally, using the fact that the average analytical run size is $\Omega(B)$, we can obtain a bound on the number of runs of 0s, 1s, and 2s in the witness transcript T. (Here, we are using the term "run" in the string sense, e.g., a run of 0s is a maximal sequence of consecutive zeros in the string.)

▶ **Lemma 17.** *Consider the execution of* BuildWitnesses$(D, h(u), \ell)$ *for some* $\ell > 0$ *and suppose that the analytical runs that are created are not all adjacent. Then the resulting witness transcript* T *is a trinary string that has at most* $O(|T|/B)$ *runs of 0s, 1s, and 2s.*

Proof. Let $s_1, s_2, \ldots, s_j$ be the sizes of the analytical runs that we add to `Unprocessed` during the execution of the algorithm. The total number of 1s and 0s in T is exactly $\sum_i s_j$, and the total number of 2s in T is also exactly $\sum_i s_j$, so $|T| = 2\sum_i s_j$. On the other hand, both the number of 1s in T and the number of *runs* of 0s in T are bounded by j. It follows that the total number of runs of 0s and 1s in T is at most $2j$, which by Lemma 16 is at most $O(\sum_i s_i/B) = O(|T|/B)$. The number of runs of 2s is at most one greater than the number of runs of 0s and 1s (this is true for any trinary string), so the total number of runs overall in T is at most $O(|T|/B)$. ◀

The fact that the witness transcript T has so few runs will lead to a much smaller bound on the number of options for T than we had in the non-chunked setting. This is the key insight that makes it possible to prove Theorem 2. As noted earlier, the proof actually separates into two cases, the case where the analytical runs are all adjacent (this case contributes $O(q^2)$ to the running time) and the case where they are not (this case contributes only $O(1)$ to the running time, and is where we make use of the previous lemmas).

Proof of Theorem 2. Suppose the insertion takes time $t \geq 0$, and let us bound $\Pr[t > \ell]$ for some ℓ. Consider the execution of BuildWitnesses$(D, h(u), \ell)$, producing witness set S and witness transcript T. Let $r_1, r_2, \ldots$ be the analytical runs produced by the algorithm. Let $\mathcal{C}$ be the indicator random variable for the event that $r_1, r_2, \ldots$ are all adjacent, and let $\overline{\mathcal{C}}$ be the indicator random variable for the event that they are not all adjacent.

We begin by considering the event $\mathcal{C} = 1$. In this case, the witness set S is a contiguous interval $[i, j]$ containing $h(u)$. By Lemma 5, if $t > \ell$, then at least $|S| \geq \ell$ elements y in the hash table have hashes $h(y) \in V$. From the standard analysis of linear-probing hash tables (e.g., Lemma 15 of [9]), we know that the probability of any such interval S existing is at most $2^{-\Omega(\ell/q^2)}$. We have therefore shown that

$$\Pr[t\mathcal{C} > \ell] \leq 2^{-\Omega(\ell/q^2)}.$$

So, $\mathbb{E}[t\mathcal{C}] = O(q^2)$ and for any $j \geq 1$, $\Pr[t\mathcal{C} > jq^2] \leq 2^{-\Omega(j)}$.

Now, for the rest of the proof, consider the event $\overline{\mathcal{C}}$. Define $\overline{t} = t\overline{\mathcal{C}}$. We will complete the proof by showing that $\mathbb{E}[\overline{t}] = O(1)$ and that for any $j \geq 1$, $\Pr[\overline{t} > jq^2] \leq 2^{-\Omega(j)}$.

Supposing that $\overline{\mathcal{C}}$ occurs, we have by Lemma 16 that $|T| \geq B$. We also have trivially that $|T| \geq \ell$, so $|T| \geq \max(B, \ell)$. The fact that we only need to consider $|T| \geq \max(B, \ell)$ will be important through the rest of the proof.

By Lemma 17, T has at most $O(|T|/B)$ runs. By standard counting arguments, the number of trinary strings of length ℓ' with $O(\ell'/B)$ runs is at most $B^{O(\ell'/B)}$. It follows, that for a given length $\ell' > \max(B, \ell)$ for T, the number of options for what T could be is at most $B^{O(\ell'/B)}$. Since the witness set S is fully determined by T, and is exactly half of T's size (Lemma 7), the number of options for S of a given size $\ell'' \geq \max(B, \ell)/2$ is at most $B^{O(\ell''/B)}$.

On the other hand, if the insertion took time more than ℓ, then, in order for a given option for S to occur, there would need to be at least $|S|$ elements y in the hash table satisfying $h(y) \in S$ (Lemma 5). For a given set S, the expected number of elements y satisfying $h(y) \in S$ is

$$\alpha|S| \leq (1 - 1/(c\sqrt{B/\log B}))|S| = |S| - \frac{\sqrt{S \log B}}{\sqrt{B}\alpha}\sqrt{|S|}.$$

By a Chernoff bound, the probability of a given candidate witness set S of size $\ell'' \geq \max(B, \ell)/2$ having ℓ'' elements hash to it is at most

$$2^{-\Omega(|S| \log B/(B\alpha^2))} = B^{-\Omega(\ell''/B)/\alpha^2}.$$

Taking a union bound over all $B^{O(\ell''/B)}$ options for S, the probability that any S of size $\ell'' \geq \max(B, \ell)/2$ occurs is at most

$$B^{O(\ell''/B)} B^{-\Omega(\ell''/B)/\alpha^2}.$$

Setting α to be a sufficiently small positive constant, this is at most

$$1/B^{10\ell''/B}.$$

Summing over all $\ell'' \geq \max(B,\ell)/2$, the probability that BuildWitnesses$(D, h(u), \ell)$ produces a saturated witness set S of size at least $\max(B,\ell)/2$ is at most

$$\sum_{\ell'' \geq \max(B,\ell)/2} 1/B^{10\ell''/B} \leq 1/B^{\max(B,\ell)/B}.$$

But, the only way that $\bar{t} \geq \ell$ can occur is if such a witness set S is produced. Thus

$$\Pr[\bar{t} \geq \ell] \leq 1/B^{\max(B,\ell)/B} = B^{B/2+\ell/(2B)} \leq q^{-\Omega(q^2 \log q + \ell/(q^2 \log q))} \leq q^{-\Omega(q^2 \log q)} \cdot 2^{-\Omega(\ell/q^2)}.$$

It follows that $\mathbb{E}[\bar{t}] \leq O(1)$, and that, for $j \geq 1$, $\Pr[\bar{t} > jq^2] \leq 2^{-\Omega(j)}$, as desired. ◀

References

1. Abseil, 2017. Accessed: 2020-11-06. URL: https://abseil.io/.
2. Ole Amble and Donald Ervin Knuth. Ordered hash tables. *The Computer Journal*, 17(2):135–142, January 1974. doi:10.1093/comjnl/17.2.135.
3. Yuriy Arbitman, Moni Naor, and Gil Segev. De-amortized cuckoo hashing: Provable worst-case performance and experimental results. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming (ICALP 2009)*, volume 5555 of *Lecture Notes in Computer Science*, pages 107–118, 2009. doi:10.1007/978-3-642-02927-1_11.
4. Guy de Balbine. *Computational analysis of the random components induced by a binary equivalence relation*. PhD thesis, California Institute of Technology, 1968.
5. Vladimir Batagelj. The quadratic hash method when the table size is not a prime number. *Communications of the ACM*, 18(4):216–217, 1975.
6. Daniel Bauer. Columbia COMS W3134: Data structures in Java — Lecture 12: Introduction to hashing, October 2015. URL: http://www.cs.columbia.edu/~bauer/cs3134-f15/slides/w3134-1-lecture12.pdf.
7. James R Bell and Charles H Kaman. The linear quotient hash code. *Communications of the ACM*, 13(11):675–676, 1970.
8. Michael A Bender, Martín Farach-Colton, John Kuszmaul, William Kuszmaul, and Mingmou Liu. On the optimal time/space tradeoff for hash tables. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1284–1297, 2022.
9. Michael A Bender, Bradley C Kuszmaul, and William Kuszmaul. Linear probing revisited: Tombstones mark the demise of primary clustering. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1171–1182. IEEE, 2022.
10. Pedro Celis, Per-Åke Larson, and J. Ian Munro. Robin Hood hashing (preliminary report). In *26th Annual Symposium on Foundations of Computer Science (FOCS'85)*, pages 281–288, Portland, Oregon, USA, 21–23 October 1985. doi:10.1109/SFCS.1985.48.
11. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts, USA, 3rd edition, 2009.
12. Lilian de Greef. UW CSE 373: Data structures and algorithims — Lecture 7: Hash table collisions, summer 2017. URL: https://courses.cs.washington.edu/courses/cse373/17su/lectures/Lecture%2007%20-%20Hash%20Table%20Collisions.pdf.
13. Erik D. Demaine, Friedhelm Meyer auf der Heide, Rasmus Pagh, and Mihai Pătraşcu. De dictionariis dynamicis pauco spatio utentibus (*lat.* on dynamic dictionaries using little space). In *Proceedings of the 7th Latin American Symposium on Theoretical Informatics (LATIN 2006)*, volume 3887 of *Lecture Notes in Computer Science*, pages 349–361, Valdiva, Chile, 20–24 March 2006. doi:10.1007/11682462_34.
14. Martin Dietzfelbinger, Andreas Goerdt, Michael Mitzenmacher, Andrea Montanari, Rasmus Pagh, and Michael Rink. Tight thresholds for cuckoo hashing via XORSAT. In *37th International Colloquium on Automata, Languages and Programming (ICALP 2010)*, pages 213–225, 2010.

15 Adam Drozdek and Donald L. Simon. *Data Structures in C*. PWS, Boston, Massachusetts, USA, 1995.

16 A Ecker. The period of search for the quadratic and related hash methods. *The Computer Journal*, 17(4):340–343, 1974.

17 Nikolaos Fountoulakis, Megha Khosla, and Konstantinos Panagiotou. The multiple-orientability thresholds for random hypergraphs. *Combinatorics, Probability and Computing*, 25(6):870–908, 2016.

18 David Gries and Doug James. Cornell CS210: Object-oriented programming and data structures – recitation week 8: Hashing, fall 2014. URL: `https://www.cs.cornell.edu/courses/cs2110/2014fa/recitations/recitation08/HashPresentation.pptx`.

19 Leo J Guibas and Endre Szemeredi. The analysis of double hashing. In *Proceedings of the eighth annual ACM symposium on Theory of computing*, pages 187–191, 1976.

20 Leo J Guibas and Endre Szemeredi. The analysis of double hashing. *Journal of Computer and System Sciences*, 16(2):226–274, 1978.

21 Leonidas J Guibas. *The analysis of hashing algorithms*. PhD thesis, Stanford University., 1976.

22 F Robert A Hopgood and J Davenport. The quadratic hash method when the table size is a power of 2. *The Computer Journal*, 15(4):314–315, 1972.

23 Gregory Kesden. CMU 15-310: System-level software development — hashing review, 2007. Accessed 31-May-2021. URL: `https://www.andrew.cmu.edu/course/15-310/applications/ln/hashing-review.html`.

24 Donald E Knuth. Notes on "open" addressing. *Unpublished memorandum*, pages 11–97, 1963.

25 Donald E Knuth. *The Art of Computer Programming: Volume 3: Sorting and Searching*. Addison-Wesley Professional, 1998.

26 Alan G Konheim and Benjamin Weiss. An occupancy discipline and applications. *SIAM Journal on Applied Mathematics*, 14(6):1266–1274, 1966.

27 Tianxiao Li, Jingxun Liang, Huacheng Yu, and Renfei Zhou. Tight cell-probe lower bounds for dynamic succinct dictionaries. *arXiv preprint arXiv:2306.02253*, 2023.

28 Tianxiao Li, Jingxun Liang, Huacheng Yu, and Renfei Zhou. Dynamic dictionary with subconstant wasted bits per key. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 171–207. SIAM, 2024.

29 Mingmou Liu, Yitong Yin, and Huacheng Yu. Succinct filters for sets of unknown sizes. In *Proceedings 47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, volume 168 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 79:1–79:19, Saarbrücken, Germany, 8–11 July 2020. `doi:10.4230/LIPIcs.ICALP.2020.79`.

30 George Lueker and Mariko Molodowitch. More analysis of double hashing. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 354–359, 1988.

31 George S Lueker and Mariko Molodowitch. More analysis of double hashing. *Combinatorica*, 13(1):83–96, 1993.

32 Ward Douglas Maurer. Programming technique: An improved hash code for scatter storage. *Communications of the ACM*, 11(1):35–38, 1968.

33 Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*. Cambridge university press, 2017.

34 Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *Journal of Algorithms*, 51(2):122–144, 2004.

35 W Wesley Peterson. Addressing for random-access storage. *IBM journal of Research and Development*, 1(2):130–146, 1957.

36 Charles E Radke. The use of quadratic residue research. *Communications of the ACM*, 13(2):103–105, 1970.

37 Rajeev Raman and Satti Srinivasa Rao. Succinct dynamic dictionaries and trees. In *Automata, Languages and Programming*, pages 357–368, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

38 Stefan Richter, Victor Alvarez, and Jens Dittrich. A seven-dimensional analysis of hashing methods and its implications on query processing. *PVLDB*, 9(3):96–107, 2015.

39 David G. Sullivan. Harvard CS S-111: Intensive introduction to computer science using Java – unit 9, part 4: Hash tables, summer 2021. URL: `https://sites.fas.harvard.edu/~libs111/files/lectures/unit9-4.pdf`.

40 Mark Allen Weiss. *Data Structures and Problem Solving using C++*. Addison-Wesley, Reading, Massachusetts, USA, 2000.