Lecture Notes on

Linear Probing with 5-Independent Hashing

Mikkel Thorup

September 16, 2015

Abstract

These lecture notes show that linear probing takes expected constant time if the hash function is 5-independent. This result was first proved by Pagh et al. [STOC'07,SICOMP'09]. The simple proof here is essentially taken from [Pătraşcu and Thorup ICALP'10]. The lecture is a nice illustration of the use of higher moments in data structures, and could be used in a course on randomized algorithms.

1 k-independence

The concept of k-independence was introduced by Wegman and Carter [34] in FOCS'79 and has been the cornerstone of our understanding of hash functions ever since. Formally, we think of a hash function $h : [u] \to [t]$ as a random variable distributed over $[t]^{[u]}$. Here $[s] = \{0, \dots, s-1\}$. We say that h is *k-independent* if for any distinct keys $x_0, \dots, x_{k-1} \in [u]$ and (possibly non-distinct) hash values $y_0, \dots, y_{k-1} \in [t]$, we have $\Pr[h(x_0) = y_0 \wedge \cdots \wedge h(x_{k-1}) = y_{k-1}] = 1/t^k$. Equivalently, we can define k-independence via two separate conditions; namely,

(a) for any distinct keys $x_0, \dots, x_{k-1} \in [u]$, the hash values $h(x_0), \dots, h(x_{k-1})$ are independent random variables, that is, for any (possibly non-distinct) hash values $y_0, \dots, y_{k-1} \in [t]$ and $i \in [k]$, $\Pr[h(x_i) = y_i] = \Pr\left[h(x_i) = y_i \mid \bigwedge_{j \in [k] \setminus \{i\}} h(x_j) = y_j\right]$, and

(b) for any $x \in [u]$, $h(x)$ is uniformly distributed in $[t]$.

As the concept of independence is fundamental to probabilistic analysis, k-independent hash functions are both natural and powerful in algorithm analysis. They allow us to replace the heuristic assumption of truly random hash functions that are uniformly distributed in $[t]^{[u]}$, hence needing $u \lg t$ random bits ($\lg = \log_2$), with real implementable hash functions that are still "independent enough" to yield provable performance guarantees similar to those proved with true randomness. We are then left with the natural goal of understanding the independence required by algorithms.

Once we have proved that k-independence suffices for a hashing-based randomized algorithm, we are free to use *any* k-independent hash function. The canonical construction of a k-independent hash function is based on polynomials of degree $k-1$. Let $p \geq u$ be prime. Picking random $a_0, \dots, a_{k-1} \in [p] = \{0, \dots, p-1\}$, the hash function is defined by:

$$h(x) = \left(a_{k-1}x^{k-1} + \cdots + a_1 x + a_0\right) \bmod p \tag{1}$$

If we want to limit the range of hash values to $[t]$, we use $h'(x) = h(x) \bmod t$. This preserves requirement (a) of independence among k hash values. Requirement (b) of uniformity is close to satisfied if $p \gg t$. More precisely, for any key $x \in [p]$ and hash value $y \in [t]$, we get $1/t - 1/p < \Pr[h'(x) = y] < 1/t + 1/p$.

Sometimes 2-independence suffices. For example, 2-independence implies so-called universality [7]; namely that the probability of two keys x and y colliding with $h(x) = h(y)$ is $1/t$; or close to $1/t$ if the uniformity of (b) is only approximate. Universality implies expected constant time performance of hash tables implemented with chaining. Universality also suffices for the 2-level hashing of Fredman et al. [14], yielding static hash tables with constant query time.

At the other end of the spectrum, when dealing with problems involving n objects, $O(\lg n)$-independence suffices in a vast majority of applications. One reason for this is the Chernoff bounds of [29] for k-independent events, whose probability bounds differ from the full-independence Chernoff bound by $2^{-\Omega(k)}$. Another reason is that random graphs with $O(\lg n)$-independent edges [2] share many of the properties of truly random graphs.

The independence measure has long been central to the study of randomized algorithms. It applies not only to hash functions, but also to pseudo-random number generators viewed as assigning hash values to $0, 1, 2, \ldots$. For example, [18] considers variants of QuickSort, [1] consider the maximal bucket size for hashing with chaining, and [17, 12] consider Cuckoo hashing. In several cases [1, 12, 18], it is proved that linear transformations $x \mapsto \big((ax + b) \bmod p\big)$ do not suffice for good performance, hence that 2-independence is not in itself sufficient.

Our focus in these notes is linear probing described below.

2 Linear probing

Linear probing is a classic implementation of hash tables. It uses a hash function h to map a set of n keys into an array of size t. When inserting x, if the desired location $h(x) \in [t]$ is already occupied, the algorithm scans $h(x)+1, h(x)+2, \ldots, t-1, 0, 1, \ldots$ until an empty location is found, and places x there. Below, for simplicity, we ignore the wrap-around from $t-1$ to 0, so a key x is always placed in a location $i \geq h(x)$.

To search a key x, the query algorithm starts at $h(x)$ and scans either until it finds x, or runs into an empty position, which certifies that x is not in the hash table. When the query search is unsuccessful, that is, when x is not stored, the query algorithm scans exactly the same locations as an insert of x. A general bound on the query time is hence also a bound on the insertion time.

Deletions are slightly more complicated. The invariant we want to preserve is that if a key x is stored at some location $i \in [t]$, then all locations from $h(x)$ to i are filled; for otherwise the above search would not get to x. Suppose now that x is deleted from location i. We then scan locations $j = i+1, i+2, \ldots$ for a key y with $h(y) \leq i$. If such a y is found at location j, we move y to location i, but then, recursively, we have to try refilling j, looking for a later key z with $h(z) \leq j$. The deletion process terminates when we reach an empty location d, for then the invariant says that there cannot be a key y at a location $j > d$ with $h(y) \leq d$. The recursive refillings always visit successive locations, so the total time spent on deleting x is proportional to the number of locations from that of x and to the first empty location. Summing up, we have

Theorem 1 *With linear probing, the time it takes to search, insert, or delete a key x is at most proportional to the number of locations from $h(x)$ to the first empty location.*

Independence	2	3	4	≥ 5
Query time	$\Theta(\sqrt{n})$	$\Theta(\lg n)$	$\Theta(\lg n)$	$\Theta(1)$
Construction time	$\Theta(n \lg n)$	$\Theta(n \lg n)$	$\Theta(n)$	$\Theta(n)$

Table 1: Expected time bounds for linear probing with a poor k-independent hash function. The bounds are worst-case expected, e.g., a lower bound for the query means that there is a concrete combination of stored set, query key, and k-independent hash function with this expected search time while the upper-bound means that this is the worst expected time for any such combination. Construction time refers to the worst-case expected total time for inserting n keys starting from an empty table.

With n the nunber of keys and t the size of the table, we call n/t the *load* of our table. We generally assume that the load is bounded from 1, e.g., that the number of keys is $n \leq \frac{2}{3}t$. With a good distribution of keys, we would then hope that the number of locations from $h(x)$ to an empty location is $O(1)$.

This classic data structure is one of the most popular implementations of hash tables, due to its unmatched simplicity and efficiency. The practical use of linear probing dates back at least to 1954 to an assembly program by Samuel, Amdahl, Boehme (c.f. [21]). On modern architectures, access to memory is done in cache lines (of much more than a word), so inspecting a few consecutive values is typically only slightly worse that a single memory access. Even if the scan straddles a cache line, the behavior will still be better than a second random memory access on architectures with prefetching. Empirical evaluations [4, 15, 24] confirm the practical advantage of linear probing over other known schemes, e.g., chaining, but caution [15, 33] that it behaves quite unreliably with weak hash functions. Taken together, these findings form a strong motivation for theoretical analysis.

Linear probing was shown to take expected constant time for any operation in 1963 by Knuth [20], in a report which is now regarded as the birth of algorithm analysis. This analysis, however, assumed a truly random hash function.

A central open question of Wegman and Carter [34] was how linear probing behaves with k-independence. Siegel and Schmidt [28, 30] showed that $O(\lg n)$-independence suffices for any operation to take expected constant time. Pagh et al. [23] showed that just 5-independence suffices for expected constant operation time. They also showed that linear transformations do not suffice, hence that 2-independence is not in itself sufficient.

Pătraşcu and Thorup [25] proved that 4-independence is not in itself sufficient for expected constant operation time. They display a concrete combination of keys and a 4-independent random hash function where searching certain keys takes super constant expected time. This shows that the 5-independence result of Pagh et al. [23] is best possible. In fact [25] provided a complete understanding of linear probing with low independence as summarized in Table 1.

Considering loads close to 1, that is load $(1 - \varepsilon)$, Pătraşcu and Thorup [26] proved that the expected operation time is $O(1/\varepsilon^2)$ with 5-independent hashing, matching the bound of Knuth [20] assuming true randomness. The analysis from [26] also works for something called simple tabulation hashing that is not 5-independent, but has some other strong probabilistic properties.

3 Linear probing with 5-independence

Below we present the simplified version of the proof from [26] of the result from [23] that 5-independent hashing suffices for expected constant time with linear probing. For simplicity, we assume that the load is at most $\frac{2}{3}$. Thus we study a set S of n keys stored in a linear probing table of size $t \geq \frac{3}{2}n$. We assume that t is a power of two.

A crucial concept is a *run* R which is a maximal interval of filled positions. We have an empty position before R, which means that all keys $x \in S$ landing in R must also hash into R in the sense that $h(x) \in R$. Also, we must have exactly $r = |R|$ keys hashing to R since the position after R is empty.

By Theorem 1 the time it takes for any operation on a key q is at most proportional to the number of locations from $h(x)$ to the first empty location. We upper bound this number by $r + 1$ where r is the length of the run containing $h(q)$. Here $r = 0$ if the location $h(q)$ is empty. We note that the query key q might itself be in R, and hence be part of the run, e.g., in the case of deletions.

We want to give an expected upper bound on r. In order to limit the number of different events leading to a long run, we focus on dyadic intervals: a *(dyadic) ℓ-interval* is an interval of length 2^ℓ of the form $[i2^\ell, (i+1)2^\ell)$ where $i \in [t/2^\ell]$. Assuming that the hashing maps S uniformly into $[t]$, we expect $n2^\ell/t \leq \frac{2}{3}2^\ell$ keys to hash into a given ℓ-interval I. We say that I is "near-full" if at least $\frac{3}{4}2^\ell$ keys from $S \setminus \{q\}$ hash into I. We claim that a long run implies that some dyadic interval of similar size is near-full. More precisely,

Lemma 2 *Consider a run R of length $r \geq 2^{\ell+2}$. Then one of the first four ℓ-intervals intersecting R must be near-full.*

Proof Let $I_0, \ldots, I_3$ be the first four ℓ-intervals intersecting R. Then I_0 may only have its last end-point in R while $I_1, \ldots, I_3$ are contained in R since $r \geq 4 \cdot 2^\ell$. In particular, this means that $L = \left(\bigcup_{i \in [4]} I_i\right) \cap R$ has length at least $3 \cdot 2^\ell + 1$.

But L is a prefix of R, so all keys landing in L must hash into L. Since L is full, we must have at least $3 \cdot 2^\ell + 1$ keys hashing into L. Even if this includes the query key q, then we conclude that one of our four intervals I_i must have $3 \cdot 2^\ell/4 \geq \frac{3}{4}2^\ell$ keys from $S \setminus \{q\}$ hashing into it, implying that I_i is near-full. ∎

Getting back to our original question, we are considering the run R containing the hash of the query q.

Lemma 3 *If the run containing the hash of the query key q is of length $r \in [2^{\ell+2}, 2^{\ell+3})$, then one of the following 12 consecutive ℓ-intervals is near-full: the ℓ-interval containing $h(q)$, the 8 nearest ℓ-intervals to its left, and the 3 nearest ℓ-intervals to its right.*

Proof Let R be the run containing $h(q)$. To apply Lemma 2, we want to show that the first four ℓ-intervals intersecting R has to be among the 12 mentioned in Lemma 3. Since the run R containing $h(q)$ has length less than $8 \cdot 2^\ell$, the first ℓ-interval intersecting R can be at most 8 before the one containing $h(q)$. The 3 following intervals are then trivially contained among the 12. ∎

Thus, conditioned on the hash of the query key, for each ℓ we are interested in a bound P_ℓ on the probability that any given ℓ-interval is near-full. Then the probability that the run containing $h(q)$ has length $r \in [2^{\ell+2}, 2^{\ell+3})$ is bounded by $12P_\ell$. Of course, this only gives us a bound for $r \geq 4$.

We thus conclude that the expected length of the run containing the hash of the query key q is bounded by

$$3 + \sum_{\ell=0}^{\log_2 t} 2^{\ell+3} \cdot 12 P_\ell = O\left(1 + \sum_{\ell=0}^{\log_2 t} 2^\ell P_\ell\right).$$

Combined with Theorem 1, we have now proved

Theorem 4 *Consider storing a set S of keys in a linear probing table of size t where t is a power of two. Conditioned on the hash of a key q, let P_ℓ bound the probability that $\frac{3}{4}2^\ell$ keys from $S \setminus \{q\}$ hash to any given ℓ-interval. Then the expected time to search, insert, or delete q is bounded by*

$$O\left(1 + \sum_{\ell=0}^{\log_2 t} 2^\ell P_\ell\right).$$

We note that Theorem 4 does not mention the size of S. However, as mentioned earlier, with a uniform distribution, the expected number of elements hashing to an ℓ-interval is $\leq 2^\ell |S|/t$, so for P_ℓ to be small, we want this expectation to be significantly smaller than $\frac{3}{4}2^\ell$. Assuming $|S| \leq \frac{2}{3}t$, the expected number is $\frac{2}{3}2^\ell$.

To get constant expected cost for linear probing, we are going to assume that the hash function used is 5-independent. This means that no matter the hash value $h(q)$ of q, conditioned on $h(q)$, the keys from $S \setminus \{q\}$ are hashed 4-independently. This means that if X_x is the indicator variable for a key $x \in S \setminus \{q\}$ hashing to a given interval I, then the variables $X_x,\ x \in S \setminus \{q\}$ are 4-wise independent.

3.1 Fourth moment bound

The probabilistic tool we shall use here to analyze 4-wise independent variables is a 4^{th} moment bound. For $i \in [n]$, let $X_i \in [2] = \{0,1\}$, $p_i = \Pr[X_i = 1] = \mathrm{E}[X_i]$, $X = \sum_{i \in [n]} X_i$, and $\mu = \mathrm{E}[X] = \sum_{i \in [n]} p_i$. Also $\sigma_i^2 = \mathrm{Var}[X_i] = \mathrm{E}[(X_i - p_i)^2] = p_i(1-p_i)^2 + (1-p_i)p_i^2 = p_i - p_i^2$. As long as the X_i are pairwise independent, the variance of the sum is the sum of the variances, so we define

$$\sigma^2 = \mathrm{Var}[X] = \sum_{i \in [n]} \mathrm{Var}[X_i] = \sum_{i \in [n]} \sigma_i^2 \leq \mu.$$

By Chebyshev's inequality, we have

$$\Pr[|X - \mu| \geq d\sqrt{\mu}] \leq \Pr[|X - \mu| \geq d\sigma] \leq 1/d^2. \tag{2}$$

We are going to prove a stronger bound if the variables are 4-wise independent and $\mu \geq 1$ (and which is only stronger if $d \geq 2$).

Theorem 5 *If the variables $X_0, \ldots, X_{n-1} \in \{0,1\}$ are 4-wise independent, $X = \sum_{i \in [n]} X_i$, and $\mu = \mathrm{E}[X] \geq 1$, then*

$$\Pr[|X - \mu| \geq d\sqrt{\mu}] \leq 4/d^4.$$

Proof Note that $(X - \mu) = \sum_{i \in [n]} (X_i - p_i)$. By linearity of expectation, the fourth moment is:

$$\mathrm{E}[(X-\mu)^4] = \mathrm{E}[\big(\sum_i X_i - p_i\big)^4] = \sum_{i,j,k,l \in [n]} \mathrm{E}\big[(X_i - p_i)(X_j - p_j)(X_k - p_k)(X_l - p_l)\big].$$

Consider a term $\mathrm{E}\big[(X_i - p_i)(X_j - p_j)(X_k - p_k)(X_l - p_l)\big]$. The at most 4 distinct variables are completely independent. Suppose one of them, say, X_i, appears only once. By definition, $\mathrm{E}\big[(X_i - p_i)\big] = 0$, and since it is independent of the other factors, we get $\mathrm{E}\big[(X_i - p_i)(X_j - p_j)(X_k - p_k)(X_l - p_l)\big] = 0$. We can therefore ignore all terms where any variable appears once. We may therefore assume that each variables appears either twice or 4 times. In terms with variables appearing twice, we have two indices $a < b$ where a is assigned to two of i, j, k, l, while b is assigned to the other two, yielding $\binom{4}{2}$ combinations based on $a < b$. Thus we get

$$\begin{aligned}
\mathrm{E}[(X-\mu)^4] &= \sum_{i,j,k,l \in [n]} \mathrm{E}\big[(X_i - p_i)(X_j - p_j)(X_k - p_k)(X_l - p_l)\big] \\
&= \sum_i \mathrm{E}\big[(X_i - p_i)^4\big] + \binom{4}{2} \sum_{a<b} \left(\mathrm{E}\big[(X_a - p_a)^2\big]\mathrm{E}\big[(X_b - p_b)^2\big]\right).
\end{aligned}$$

Considering any multiplicity $m = 2, 3, 4, \ldots$, we have

$$\begin{aligned}
\mathrm{E}[(X_i - p_i)^m] &= p_i(1-p_i)^m + (1-p_i)(-p_i)^m \\
&\le p_i(1-p_i)^m + (1-p_i)p_i^m \\
&= p_i(1-p_i)\left((1-p_i)^{m-1} + p_i^{m-1}\right) \\
&\le p_i(1-p_i) = \sigma_i^2. \qquad (3)
\end{aligned}$$

Thus, continuing our calculation, we get

$$\begin{aligned}
\mathrm{E}[(X-\mu)^4] &= \sum_i \mathrm{E}\big[(X_i - p_i)^4\big] + \binom{4}{2} \sum_{a<b} \left(\mathrm{E}\big[(X_a - p_a)^2\big]\mathrm{E}\big[(X_b - p_b)^2\big]\right) \\
&\le \sum_i \sigma_i^2 + \binom{4}{2} \sum_{a<b} \sigma_a^2 \sigma_b^2 \\
&\le \sigma^2 + 3\left(\sum_i \sigma_i^2\right)^2 \\
&= \sigma^2 + 3\sigma^4. \qquad (4)
\end{aligned}$$

Since $\sigma^2 \le \mu$, with $\mu \ge 1$, we get

$$\mathrm{E}[(X-\mu)^4] \le 4\mu^2.$$

Hence, by Markov's inequality,

$$\Pr[|X-\mu| \ge d\sqrt{\mu}] = \Pr[(X-\mu)^4 \ge (d\sqrt{\mu})^4] \le \mathrm{E}[(X-\mu)^4]/(d\sqrt{\mu})^4 \le 4/d^4.$$

This completes the proof of Theorem 5. ∎

We are now ready to prove the 5-independence suffices for linear probing.

Theorem 6 *Suppose we use a 5-independent hash function h to store a set S of n keys in a linear probing table of size $t \ge \frac{3}{2}n$ where t is a power of two. Then it takes expected constant time to search, insert, or delete a key.*

Proof First we fix the hash of the query key q. To apply Theorem 4, we need to find a bound P_ℓ bound on probability that $\frac{3}{4}2^\ell$ keys from $S \setminus \{q\}$ hash to any given ℓ-interval I. For each key $x \in S \setminus \{q\}$, let X_x be the indicator variable for $h(x) \in I$. Then $X = \sum_{x \in S \setminus \{q\}} X_x$ is the number of keys landing I, and the expectation of X is $\mu = \mathrm{E}[X] \leq n2^\ell/t \leq \frac{2}{3}2^\ell$. Our concern is the event that

$$X \geq \frac{3}{4}2^\ell \implies X - \mu \geq \frac{1}{12}2^\ell > \frac{1}{10}\sqrt{2^\ell \mu}.$$

Since h is 5-independent, the X_x are 4-independent, so by Theorem 5, we get

$$\Pr\left[X \geq \frac{3}{4}2^\ell\right] \leq 40000/2^{2\ell} = O(1/2^{2\ell}).$$

Thus we can use $P_\ell = O(1/2^{2\ell})$ in Theorem 4, and then we get that the expected operation cost is

$$O\left(1 + \sum_{\ell=0}^{\log_2 t} 2^\ell P_\ell\right) = O\left(1 + \sum_{\ell=0}^{\log_2 t} 2^\ell/2^{2\ell}\right) = O(1).$$

∎

Problem 1 *Above we assumed that the range of our hash function is* $[t]$ *where* t *is a power of two. As suggested in the introduction, we use a hash function based on a degree 4 polynomial over a prime field* $\mathbb{Z}_p$ *where* $p \gg 1$*, that is, we pick* 5 *independent random coefficients* $a_0, \ldots, a_4 \in [p]$*, and define the hash function* $h' : [p] \to [t]$ *by*

$$h'(x) = \Big(\big(a_4 x^4 + \cdots + a_1 x + a_0\big) \bmod p\Big) \bmod t.$$

Then for any distinct $x_0, \ldots, x_4$*, the hash values* $h'(x_0), \ldots, h'(x_4)$ *are independent. Moreover, we have almost uniformity in the sense that for any* $x \in [p]$ *and* $y \in [t]$*, we have* $1/t - 1/p < \Pr[h'(x) = y] < 1/t + 1/p$*.*

Prove that Theorem 6 still holds with constant operation time if $p \geq 24t$*.*

Problem 2 *Assuming full randomness, use Chernoff bounds to prove that the longest run in the hash table has length* $O(\log n)$ *with probability at least* $1 - 1/n^{10}$*.*

Problem 3 *Using Chebyshev's inequality, show that with 3-independent hashing, the expected operation time is* $O(\log n)$*.*

4 The k-th moment

The 4^{th} moment bound used above generalizes to any even moment. First we need

Theorem 7 *Let* $X_0, \ldots, X_{n-1} \in \{0,1\}$ *be* k*-wise independent variables for some (possibly odd)* $k \geq 2$*. Let* $p_i = \Pr[X_i = 1]$ *and* $\sigma_i^2 = \mathrm{Var}[X_i] = p_i - p_i^2$*. Moreover, let* $X = \sum_{i \in [n]} X_i$*,* $\mu = \mathrm{E}[X] = \sum_{i \in [n]} p_i$*, and* $\sigma^2 = \mathrm{Var}[X] = \sum_{i \in [n]} \sigma_i^2$*. Then*

$$\mathrm{E}[(X - \mu)^k] \leq O(\sigma^2 + \sigma^k) = O(\mu + \mu^{k/2}).$$

Proof The proof is a simple generalization of the proof of Theorem 5 up to (4). We have

$$(X-\mu)^k = \sum_{i_0,\ldots,i_{k-1}\in[n]} \big((X_{i_0}-p_{i_0})(X_{i_1}-p_{i_1})\cdots(X_{i_{k-1}}-p_{i_{k-1}})\big)$$

By linearity of expectation,

$$\mathrm{E}[(X-\mu)^k] = \sum_{i_0,\ldots,i_{k-1}\in[n]} \mathrm{E}\left[\big((X_{i_0}-p_{i_0})(X_{i_1}-p_{i_1})\cdots(X_{i_{k-1}}-p_{i_{k-1}})\big)\right]$$

We now consider a specific term

$$\big((X_{i_0}-p_{i_0})(X_{i_1}-p_{i_1})\cdots(X_{i_{k-1}}-p_{i_{k-1}})\big)$$

Let $j_0 < j_1 < \cdots < j_{c-1}$ be the distinct indices among $i_0, i_1, \ldots, i_{n-1}$, and let m_h be the multiplicity of j_h. Then

$$\begin{aligned}&\big((X_{i_0}-p_{i_0})(X_{i_1}-p_{i_1})\cdots(X_{i_{k-1}}-p_{i_{k-1}})\big)\\ &\quad= \big((X_{j_0}-p_{j_0})^{m_0}(X_{j_1}-p_{j_1})^{m_1}\cdots(X_{j_{c-1}}-p_{j_{c-1}})^{m_{c-1}}\big)\,.\end{aligned}$$

The product involves at most k different variables so they are all independent, and therefore

$$\begin{aligned}&\mathrm{E}\left[\big((X_{j_0}-p_{j_0})^{m_0}(X_{j_1}-p_{j_1})^{m_1}\cdots(X_{j_{c-1}}-p_{j_{c-1}})^{m_{c-1}}\big)\right]\\ &\quad= \mathrm{E}\left[(X_{j_0}-p_{j_0})^{m_0}\right]\,\mathrm{E}\left[(X_{j_1}-p_{j_1})^{m_1}\right]\cdots\mathrm{E}\left[(X_{j_{c-1}}-p_{j_{c-1}})^{m_{c-1}}\right]\end{aligned}$$

Now, for any $i \in [n]$, $\mathrm{E}[X_i - p_i] = 0$, so if any multiplicity is 1, the expected value is zero. We therefore only need to count terms where all multiplicities m_h are at least 2. The sum of multiplicities is $\sum_{h\in[c]} m_h = k$, so we conclude that there are $c \le k/2$ distinct indices $j_0, \ldots, j_{c-1}$. Now by (3),

$$\mathrm{E}\left[(X_{j_0}-p_{j_0})^{m_0}\right]\,\mathrm{E}\left[(X_{j_1}-p_{j_1})^{m_1}\right]\cdots\mathrm{E}\left[(X_{j_{c-1}}-p_{j_{c-1}})^{m_{c-1}}\right] \le \sigma_{j_0}^2\sigma_{j_1}^2\cdots\sigma_{j_{c-1}}^2.$$

We now want to bound the number tuples $(i_0, i_1, \ldots, i_{k-1})$ that have the same c distinct indices $j_0 < j_1 < \cdots < j_{c-1}$. A crude upper bound is that we have c choices for each i_h, hence c^k tuples.

We therefore conclude that

$$
\begin{aligned}
\mathrm{E}[(X-\mu)^k] &= \sum_{i_0,\ldots,i_{k-1}\in[n]} \mathrm{E}\left[\left((X_{i_0}-p_{i_0})(X_{i_1}-p_{i_1})\cdots(X_{i_{k-1}}-p_{i_{k-1}})\right)\right] \\
&\leq \sum_{c=1}^{\lfloor k/2\rfloor}\left(c^k \sum_{0\leq j_0<j_1<\cdots<j_{c-1}<n} \sigma_{j_0}^2\sigma_{j_1}^2\cdots\sigma_{j_{c-1}}^2\right) \\
&\leq \sum_{c=1}^{\lfloor k/2\rfloor}\left(\frac{c^k}{c!} \sum_{j_0,j_1,\ldots,j_{c-1}\in[n]} \sigma_{j_0}^2\sigma_{j_1}^2\cdots\sigma_{j_{c-1}}^2\right) \\
&\leq \sum_{c=1}^{\lfloor k/2\rfloor}\left(\frac{c^k}{c!}\left(\sum_{j\in[n]}\sigma_j^2\right)^c\right) \\
&= \sum_{c=1}^{\lfloor k/2\rfloor}\left(\frac{c^k}{c!}\sigma^{2c}\right) \\
&= O\left(\sigma^2+\sigma^k\right) = O\left(\mu+\mu^{k/2}\right).
\end{aligned}
$$

Above we used that $c, k = O(1)$ hence that, e.g., $c^k = O(1)$. This completes the proof of Theorem 7. ■

For even moments, we now get a corresponding error probability bound

Corollary 8 *Let $X_0, \ldots X_{n-1} \in \{0,1\}$ be k-wise independent variables for some even constant $k \geq 2$. Let $p_i = \Pr[X_i = 1]$ and $\sigma_i^2 = \mathrm{Var}[X_i] = p_i - p_i^2$. Moreover, let $X = \sum_{i\in[n]} X_i$, $\mu = \mathrm{E}[X] = \sum_{i\in[n]} p_i$, and $\sigma^2 = \mathrm{Var}[X] = \sum_{i\in[n]} \sigma_i^2$. If $\mu = \Omega(1)$, then*

$$\Pr[|X-\mu| \geq d\sqrt{\mu}] = O(1/d^k).$$

Proof By Theorem 7 and Markov's inequality, we get

$$
\begin{aligned}
\Pr[|X-\mu| \geq d\sqrt{\mu}] &= \Pr[(X-\mu)^k \geq d^k\mu^{k/2}] \\
&\leq \frac{\mathrm{E}[(X-\mu)^k]}{d^k\mu^{k/2}} \\
&= \frac{O\left(\mu+\mu^{\lfloor k/2\rfloor}\right)}{d^k\mu^{k/2}} \\
&= O(1/d^k).
\end{aligned}
$$

■

Problem 4 *In the proofs of this section, where and why do we need that (a) k is a constant and (b) that k is even.*

References

[1] Noga Alon, Martin Dietzfelbinger, Peter Bro Miltersen, Erez Petrank, and Gábor Tardos. Linear hash functions. *J. ACM*, 46(5):667–683, 1999.

[2] Noga Alon and Asaf Nussboim. k-wise independent random graphs. In *Proc. 49th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 813–822, 2008.

[3] Martin Aumüller, Martin Dietzfelbinger, and Philipp Woelfel. Explicit and efficient hash families suffice for cuckoo hashing with a stash. *Algorithmica*, 70(3):428–456, 2014. Announced at ESA'12.

[4] John R. Black, Charles U. Martel, and Hongbin Qi. Graph and hashing algorithms for modern architectures: Design and performance. In *Proc. 2nd International Workshop on Algorithm Engineering (WAE)*, pages 37–48, 1998.

[5] Andrei Z. Broder, Moses Charikar, Alan M. Frieze, and Michael Mitzenmacher. Min-wise independent permutations. *Journal of Computer and System Sciences*, 60(3):630–659, 2000. Announced at STOC'98.

[6] Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig. Syntactic clustering of the web. *Computer Networks*, 29:1157–1166, 1997.

[7] Larry Carter and Mark N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, 1979. Announced at STOC'77.

[8] Edith Cohen. Size-estimation framework with applications to transitive closure and reachability. *Journal of Computer and System Sciences*, 55(3):441–453, 1997. Announced at STOC'94.

[9] Søren Dahlgaard and Mikkel Thorup. Approximately minwise independence with twisted tabulation. In *Proc. 14th Scandinavian Workshop on Algorithm Theory (SWAT)*, pages 134–145, 2014.

[10] Martin Dietzfelbinger. Universal hashing and k-wise independent random variables via integer arithmetic without primes. In *Proc. 13th Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 569–580, 1996.

[11] Martin Dietzfelbinger, Torben Hagerup, Jyrki Katajainen, and Martti Penttonen. A reliable randomized algorithm for the closest-pair problem. *Journal of Algorithms*, 25(1):19–51, 1997.

[12] Martin Dietzfelbinger and Ulf Schellbach. On risks of using cuckoo hashing with simple universal hash classes. In *Proc. 20th ACM/SIAM Symposium on Discrete Algorithms (SODA)*, pages 795–804, 2009.

[13] Martin Dietzfelbinger and Philipp Woelfel. Almost random graphs with simple hash functions. In *Proc. 25th ACM Symposium on Theory of Computing (STOC)*, pages 629–638, 2003.

[14] Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with 0(1) worst case access time. *Journal of the ACM*, 31(3):538–544, 1984. Announced at FOCS'82.

[15] Gregory L. Heileman and Wenbin Luo. How caching affects hashing. In *Proc. 7th Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 141–154, 2005.

[16] Piotr Indyk. A small approximately min-wise independent family of hash functions. *Journal of Algorithms*, 38(1):84–90, 2001. Announced at SODA'99.

[17] Daniel M. Kane Jeffery S. Cohen. Bounds on the independence required for cuckoo hashing, 2009. Manuscript.

[18] Howard J. Karloff and Prabhakar Raghavan. Randomized algorithms and pseudorandom numbers. *Journal of the ACM*, 40(3):454–476, 1993.

[19] B.W. Kernighan and D.M. Ritchie. *The C Programming Language*. Prentice Hall, 2nd edition, 1988.

[20] Donald E. Knuth. Notes on open addressing. Unpublished memorandum. See `http://citeseer.ist.psu.edu/knuth63notes.html`, 1963.

[21] Donald E. Knuth. *The Art of Computer Programming, Volume III: Sorting and Searching.* Addison-Wesley, 1973.

[22] Michael Mitzenmacher and Salil P. Vadhan. Why simple hash functions work: exploiting the entropy in a data stream. In *Proc. 19th ACM/SIAM Symposium on Discrete Algorithms (SODA)*, pages 746–755, 2008.

[23] Anna Pagh, Rasmus Pagh, and Milan Ružić. Linear probing with constant independence. *SIAM Journal on Computing*, 39(3):1107–1120, 2009. Announced at STOC'07.

[24] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *Journal of Algorithms*, 51(2):122–144, 2004. Announced at ESA'01.

[25] Mihai Pătraşcu and Mikkel Thorup. On the k-independence required by linear probing and minwise independence. In *Proc. 37th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 715–726, 2010.

[26] Mihai Pătraşcu and Mikkel Thorup. The power of simple tabulation-based hashing. *Journal of the ACM*, 59(3):Article 14, 2012. Announced at STOC'11.

[27] Mihai Pătraşcu and Mikkel Thorup. Twisted tabulation hashing. In *Proc. 24th ACM/SIAM Symposium on Discrete Algorithms (SODA)*, pages 209–228, 2013.

[28] Jeanette P. Schmidt and Alan Siegel. The analysis of closed hashing under limited randomness. In *Proc. 22nd ACM Symposium on Theory of Computing (STOC)*, pages 224–234, 1990.

[29] 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. Announced at SODA'93.

[30] Alan Siegel and Jeanette P. Schmidt. Closed hashing is computable and optimally randomizable with universal hash functions. Technical Report TR1995-687, Courant Institute, New York University, 1995.

[31] Mikkel Thorup. Even strongly universal hashing is pretty fast. In *Proc. 11th ACM/SIAM Symposium on Discrete Algorithms (SODA)*, pages 496–497, 2000.

[32] Mikkel Thorup. Bottom-k and priority sampling, set similarity and subset sums with minimal independence. In *Proc. 45th ACM Symposium on Theory of Computing (STOC)*, 2013.

[33] Mikkel Thorup and Yin Zhang. Tabulation-based 5-independent hashing with applications to linear probing and second moment estimation. *SIAM Journal on Computing*, 41(2):293–331, 2012. Announced at SODA'04 and ALENEX'10.

[34] Mark N. Wegman and Larry Carter. New classes and applications of hash functions. *Journal of Computer and System Sciences*, 22(3):265–279, 1981. Announced at FOCS'79.