Seminar Advanced Algorithms and Data Structures - Student Report

The Power of Simple Tabulation Hashing

Jonas Mohler

April 17, 2018

Introduction

In this report I will try to neatly present the results of Pătraşcu and Thorup in their paper "The Power of Simple Tabulation Hashing. In order to do so, I will first introduce the Algorithm and then the mathematical concepts used to analyse it. This should provide a clear view of what we're looking at and why we're looking at it. Pătraşcu and Thorup actually provide an implementation of Simple Tabulation Hashing for which they can give good bounds with high probability. Demonstrating their approach will be the last part of the report.

Algorithm

In this section, I am going to introduce Tabulation Hashing. I will start by showing how totally random hash functions can be used to build hash tables that would be expected to run fast, while using way too much space. Then Simple Tabulation Hashing will be presented as a way to lower the usage of space and Chaining will provide a means to handle collisions.

Totally Random Hash Functions

A hash function h: {0,1,...,u-1} $->$ {0,1,..,m-1}, so a function mapping u keys to m hashes, is called totally random if:

P[h(x) = t] = 1/m , for all keys x independent of all other keys. [1]

So in words this means a hash function is totally random, if it distributes the keys uniformly in the table and its behaviour is statistically independent for any number of keys.
Now in this state our dictionary needs Θ(u lg m) space, with Θ(lg m) bits to store the hash of a key and u keys to hash, which is no real improvement compared to the O(u lg u) of a binary tree, but with Simple Tabulation we can find a solution to this problem.

Simple Tabulation Hashing

In Simple Tabulation Hashing we initialize c totally random tables T_i, i $\in$ {1,...,c}. We view keys as vectors of c equally long characters $x_1, x_2, , x_c$ and then hash each character x_i with its respective hash function $h_i = T_i[x_i]$. We then xor all these hashes to get the hash of our original key. Now instead of having to keep a map of u keys to m hashes, we keep c maps of $2^{(log_2 u)/c} = u^{1/c}$ keys to m hashes. These c tables now only take up a combined space of size $O(cu^1/c)$, making a real improvement on the O(u lg m) of one totally random hash table. [1]

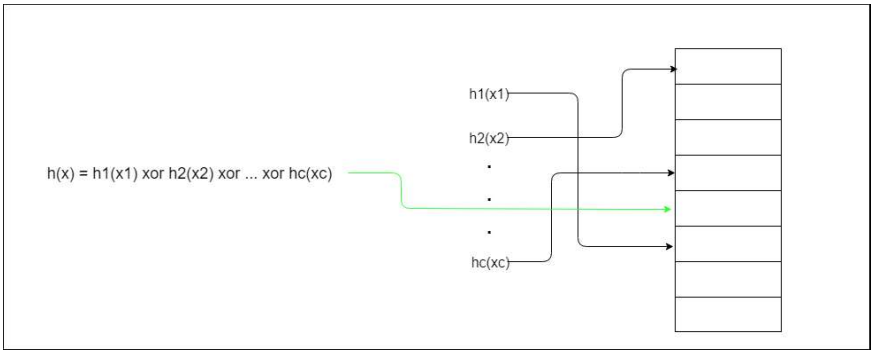

Figure 1: Simple Tabulation Hashing

Now our space complexity is reasonable, but our hash function has changed. We're now calculating our hash by xor-ing c totally random hashed characters and we can not assume independence of these hashes anymore, which plays an important role in the analysis of the algorithm.

Chaining

There is a multitude of possibilities to resolve collisions in a hash table. While the paper analyses Cuckoo Hashing, Linear Probing and Chaining, I am restraining myself to the last. Chaining is fairly simple. Instead of letting our hashes point to our data directly they now point to linked-lists, also referred to as bins, whenever we want to insert into our dictionary we simply add our element to the end of the linked-list pointed to by the keys hash, and deletion is a simple matter of redirecting a pointer.

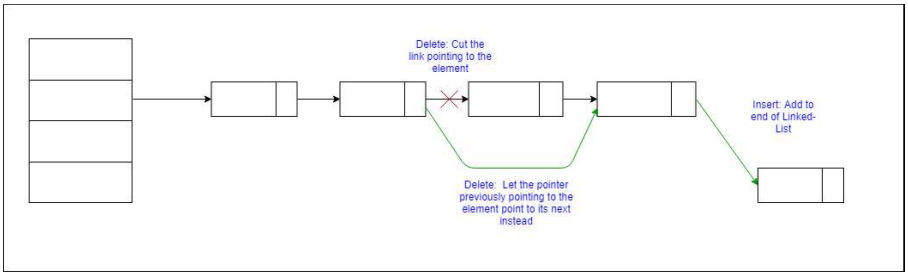

Figure 2: Example of insertion and deletion on a linked-list pointed to by a hash table

Now a totally random hash function maps keys to a *specific* linked-list with probability 1/m, so after inserting n elements we would expect that list to contain n/m of them. The hash table can always be resized on need to keep m in range of n ($m = \Theta(n)$), so n/m can be kept constant and we would expect our operations to be in O(1). [1]
Using Simple Tabulation Hashing, independence of hashes can't just be assumed, as stated above. The next section therefore attempts to provide a small insight into the methods used to analyse such a more complicated case.

Analytical Tools

When analysing hashing that makes use of families of hash functions, such as our c hash functions in simple tabulation hashing, what is usually used is k-independence. K-independence is a measure of how correlated the individual hashes are, 2-independence of a family H of hash function for example implies that for every function $h \in H$, the hash of any two distinct keys are independent random variables. K-independence, as a generalization of this concept, implies any k distinct keys are hashed independently for every $h \in H$. [1] With a certain degree of independence, depending on the implementation, we can then guarantee operations in O(1). Linear Probing for example is known to need 5-independence.
Unfortunately, Tabulation Hashing is known to only be 3-independent, which doesn't provide us with a bound strong enough to guarantee O(1) operations for Chaining. [2] Pătraşcu and Thorup are going a different way. They are using so called Chernoff Bounds to analyse the distribution of random variables, in our case the number of keys ending up in a bin, in a region of the distribution not around the mean, the so called tail. They provide their bounds "with high probability", which is understood as follows

High Probability Bound: In Probability Theory, an event E is said to occur "with high probability", if $P[E] \geq 1 - n^{-\gamma}$ for any constant γ. [1] So basically the bigger the γ we choose, the closer our bound gets to a probability of 1.

Chernoff Bounds

As stated above, Chernoff Bounds give us a way of bounding how far off the mean our random variable can be, by looking at the probability that a random variable is a prespecified constant smaller or larger than the mean. For X being the number of keys being hashed to a particular bin, the Paper will provide us with the following Bounds:

$P[X \geq (1+\delta)\mu] \leq (e^{\delta}/((1+\delta)^{1+\delta}))^{\mu/d}$ and $P[X \geq (1-\delta)\mu] \leq (e^{\delta}/((1-\delta)^{1-\delta}))^{\mu/d}$ [2]

In these bounds, μ = n/m is our mean, δ is the parameter that allows us to specify how close to the mean we are going to look, and d is a bound on the contribution of a Group (we're going to introduce these later),to a specific bin. Now as with k-independence, Chernoff Bounds require their random variables to be independent, but as it turns out not as independent as k-indepence would need. Specifically, even if our random variables depend on the previous random variables, as long as the mean of a random variable X_i, given all previous random variables, is itself a constant independent of all the previous X, our bounds hold. [2]
Showing how to guarantee this condition is basically what Patrascu and Thorup are doing and how they're doing that will be the topic of the last section.

The Papers Approach

So at this point we know how hashes are obtained in Simple Tabulation Hashing. Also, we know that if the mean of our random variable (the number of keys being hashed to a specific bin) is a constant independent of the random variables, we can apply Chernoff Bounds and thereby get a good estimation on the load of the bins. We are going to need one key Lemma to display the method.

Lemma: When hashing $n \leq m^{1-\epsilon}$ keys into m bins, for some constant $\epsilon > 0$, for all constants γ all bins get less than $d = min\{((1+\gamma)/\epsilon)^c, 2^{(1+\gamma)/\epsilon}\}$ keys, with high probability. [2]

The γ is again our parameter for high probability, and the ϵ is used to state that we hash into significantly more bins than the number of keys.
This Lemma relies on two core claims, the first of them being that any set T of keys contains a subset U of $log_2|T|$ keys, that hash independently. Specifically, if |T| $\geq$ d, then |U| $\geq (1+\gamma)/\epsilon$.
The detailed proof of this claim will not be given, but I will try to present the basic idea: We look at keys as vectors of c characters again and we define a position-character to be a character and its index in the vector it appears in. We then look at a position i in our set of keys T where we have different characters, and we pick a key x using the character that is least common. We then remove all other keys using that position-character from our set, then the key

we picked hashes independently of the hash of the remaining set of keys, since only the hash of key x depends on the hash of that position-character. So in this manner we can find an independent subset of keys for every set of keys. We define a Group G_α to be the Set containing all keys x that contain the least common position-character α. If we look at he contribution of a group to a bin as a random variable X, we know the distribution of X to depend on previous hashes, but we know the mean to be independent. Also, we know that all these X are bounded by d, so X $\leq$ d. The second claim is that the probability of having $u = (1 + \gamma)/\epsilon$ keys that independently hash to the same bin is $m^{-\gamma}$. The proof of this claim is a combinatorial argument, that can now rely on the independence of the keys.

Algorithm 1 Recursive method to hash a set S of keys while constructing independent subsets

```
1:  procedure ConstructSubsetsFrom(Set S)
2:
3:      α ← least common position-character in S;
4:
5:      G_α ← keys in S for which α is the last position-character to be fixed;
6:
7:      if there is elements in S \ G_α then
8:
9:          ConstructSubsetsFrom( S \ G_α);
10:         hash all position-characters in S except α;
11:
12:     hash α;
```

So to finally prove constant complexity of Chaining in Simple Tabulation Hashing, we go along as follows:
First, we partition the set of keys T into independent groups G by applying the algorithm above.
Using another small Lemma, relying on the fact that when constructing the Groups G at each point in time we pick our next α such that G_α is minimal, we know that with high probability, each group G contributes with $\leq$ d to each bin. [2]
For X_α being the contribution of G_α to a fixed bin, we define $X'_G = X_G$ if G_α is d-bounded and $X'_G = G_\alpha$/m (a constant) otherwise. Since with probability 1 - $m^{-\gamma}$ all groups are d-bounded and $\Sigma_\alpha X'_\alpha = \Sigma_\alpha X_\alpha$ in these cases, a probabilistic bound on X'_α is also a bound on X_α up to a constant factor. [2]
We know $E[X'_G]$ to be a constant independent on previous hashes and can therefore apply Chernoff Bounds to derive a constant load on the bins, given we keep the tables large enough, and therefore constant time for Chaining.

Conclusion

In Conclusion, the paper shows that we can get constant time complexitiy on a hashing scheme liked for its simplicity, by taking a less common approach on the analysis of such algorithms. Instead of trying to prove high independence the authors can show that by careful construction, which is essentially hashing the common characters first, a different kind of independence can be guaranteed and, using high probability bounds, this suffices to provide our hoped for bounds when using Chaining.

References

[1] MIT, Advanced Data Structures Course, Lecture Notes, Spring 2014
`https://courses.csail.mit.edu/6.851/spring14/scribe/lec10.pdf`

[2] M. Pătraşcu, M. Thorup. The Power of Simple Tabulation Hashing STOC 2011