CS 388R: Randomized Algorithms Fall 2015

Lecture 10 - Oct. 5, 2015

Prof. Eric Price *Scribe: Vivek Pradhan, Fu Li*

1 Recap Cuckoo Hashing

In the last lecture, we covered Cuckoo Hashing, which has many good properties: expected $O(1)$ time for insertion, worst case $O(1)$ time for lookup and delete. However, to construct the such good hash function, we "cheated" ourselves for assuming that all involved hash functions h were fully independent, which is hard to be satisfied in the real world.

To avoid the fully independence assumption, we show, instead of fully independence, $O(\log n)$-wise independence suffices. Recall that, in the fully independent case, the main point is to show no component has size lager than t with high probability. For $O(\log n)$-wise independent case, it is suffice for us to show the same statement still be true. That is, we need to show there are no circles in the Cuckoo graph with probability $(1 - 1/n)$ when we use $O(\log n)$-wise independent hashing function, which is easy to figure out.

For the $O(\log n)$-independent hashing, there is a easy way to construct it by using $O(\log n)$-degree polynomial. However, it doesn't work. To evaluate this hash function, it costs us $O(\log n)$ time, which breaks the promise of expected constant time lookup for Cuckoo Hashing.

Besides the $O(\log n)$-independent hashing, we can also try to use other simpler hash functions. However, it is not known whether there are simple hash functions which could work for Cuckoo Hashing. Until now, people only know there exists a 5-wise independent hashing which cannot work.

2 Perfect Hashing

Note that, the Cuckoo Hashing still use $O(n)$ spaces to store the hash value. In the following, we want to minimize the size of the hash range. In this case, we focus on how to construct a hash function with constant-time lookup and the perfect (minimal) range, without concerning the insertion or deletion. Formally, we introduce the following definition:

Definition 1 (Perfect Hashing)**.** *Suppose we have a large universe U of keys. For a n-sized set $S \subset U$, a perfect hash function $F : S \to [m]$ is a hash function that maps distinct elements in S to a set of integers, with no collisions.*

Definition 2 (Minimal Perfect Hashing Function (MPHF))**.** *For a n-sized set S, a minimal perfect hashing function is a perfect hash function $F : S \to [m]$ where $m = n$.*

Main Goal: $O(n)$ time and $O(n)$ space to construct the MPHF and can be evaluated in $O(1)$ time.

In the following, we first try to show how to construct the PHF in $O(n)$ time and $O(n)$ space with $O(1)$ lookup time.

Possible ways:

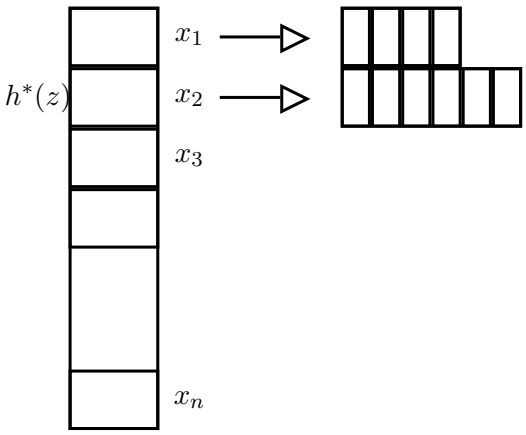
1. Use the identity function, namely $h(x) = x$, then it will use $m = U$ size.
2. List, which is very slow for lookup
3. Use 2-wise independent hashing over $2n^2$.

To analyze the last method, there are $\binom{n}{2}$ pairs and each pair collides with probability $1/m$. Thus, by union bound, we know $Pr_{h\sim H}[\exists \text{ collisions}] \leq \binom{n}{2}/m \leq 1/4$. That is, $Pr_{h\sim H}[\text{no collisions}] \geq 3/4$.

Thus we can construct the hashing with an expected constant number of hash functions. We can check each hash function in linear time, for expected linear construction time total.

But how can we decrease from $O(n^2)$ to $O(n)$ space?

2.1 Constructing Perfect Hash



We first use a 2-wise independent hash function and hash into n buckets. Then we take all the collisions and hash it into set of buckets of size $2x_i^2$ where x_i is number of collisions in that bucket. Note that each row will have length $2x_i^2$ so as not to have any collisions as shown in the figure.
Let $Y_i = \sum_{j=1}^{i-1} 2x_j^2 \Rightarrow$ our hash function is evaluated as follows (this just converts the outputs from n different hash function into sequential values):

$$h(z) = Y_h^*(z) + h_{h^*(z)}(z)$$

Let use analyze the space used to store the hash functions h_i. Note that each h_i uses $2x_i^2$ buckets. This means that space used to store the n hash functions h_i is space$(h_i) = \sum_i^n 2x_i^2$

$$\sum_i^n 2x_i^2 = 2\sum_i^n \binom{x_i}{2} + \sum_i^n x_i = 2(\text{No. of Collisions}) + n$$

$$E\left[\sum_i^n 2x_i^2\right] = 2\left(\binom{n}{2}\frac{1}{n}\right) + n \leq \frac{n^2}{n} + n = 2n.$$

Hence the total number of buckets is at most $4n$ with $1/2$ probability. Hence overall space usage is $O(n)$ for storing the mapping Y_i, the n hash functions h_i, h^*, and the contents of the buckets.

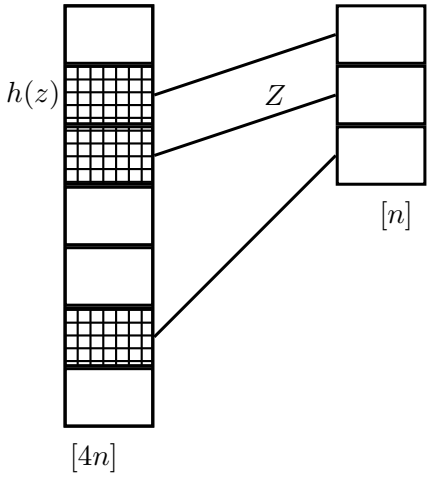
To construct the hash function, we first choose h^* repeatedly until we find one with $\sum 2x_i^2 \leq 4n$. We then choose the individual h_i until they have zero collisions. Each hash function is chosen a constant number of times in expectation, so the total construction time is $O(n)$ in expectation.

2.2 Using Perfect Hash to construct a MPHF

We now have a Perfect Hash $h : S \to [4n]$. Our goal now is to construct a minimal perfect hash $h' : S \to [n]$. The figure shows that h. Only n of the $4n$ buckets are occupied. We just construct a mapping of these occupied buckets to the set $[n]$. Let this mapping be Z. Our MPHF is simply,

$$h'(x) = Z_{h(x)}$$

. The total space is $O(n)$ to store h and $O(n)$ to store Z, for $O(n)$ total.



References

[MU05] Michael Mitzenmacher, Eli Upfal. Probability and Computing: Randomized Algorithms and Probabilistic Analysis *Cambridge University Press*, 2005.