COMS E6998: Advanced Data Structures **Spring 2019**

Lecture 10: Dictionary and Hashtable

Instructor: Omri Weinstein *Scribes: Shikun Wang, Yimin Hu*

1 Overview

In this lecture, we discussed the definition of the dictionary and different approaches to hashing. We talked about different hash functions and their properties including basic properties, k-wise independence. We talked about different approaches to using hash functions in a data structure. The approaches we cover are basic chaining and perfect hashing. The linear probing and cuckoo hashing will be covered in the next lecture.

2 Application

Data structure for information retrieval and string problems. Examples are

1. Pattern matching in text or DNA strings, file management/organizations.
2. String problems arising in large-scale storage Applications.

The main thing to distinguish is $O(1)$, at most $O(log(n))$ query time and as close as to the theoretical bound on space which is linear or $O((1+\epsilon)n)$ at most.

3 Basics

Most basic Data structure primitive and The dictionary problem:

Definition 1. *Preprocess/maintain a set* $|\mathbb{S}| = n$ *with keys* $x_1, x_2, .., x_n \in [U]$, $(n << U)$ *such that* $x \in \mathbb{S}$ *can be retrieved quickly.*

Claim 2. *The dictionary problem is easier than the Predecessor problem.*

3.1 Hashing

A natural solution is to use hashing. A randomized mapping $h : [U] \to [m]$, $(m << U)$ s.t. only few elements collide: min $Pr_h(h(x) = h(y))$. Suppose if we already has a good hash function h with nearly zero collisions $\subseteq \mathbb{S}$ and a cheap description, it is an efficient dictionary.

1. Store description in memory, $\forall x \in \mathbb{S}$, store x_i in address $h(x_i)$
2. $y \in \mathbb{S} \Rightarrow$ read description and go to $h(y)$
3. Analysis: $S = O(description) + O(m)$, $t = O(description)$

3.2 Truly random hashing

Definition 3. *A hash function h is truly iid random if $\forall x_i : h(x_i) \in_{iid} R$ such that $Pr_h(h(x) = h(y)) = \frac{1}{m}$ for some $y \neq x$.*

There are two big problems with this hashing:

1. The description length is too large: $O(ulog(n))$
2. The number of collisions is too large $\mathbb{E}\ [collisions \in |\mathbb{S}| = n] = O(\frac{n^2}{m})$.

In order to address these problems, we introduced the definition of k-wise independent.

Definition 4. *A 2-wise independent family of hash function $\mathbb{H}$ is defined as $\forall h \in \mathbb{H}$ with $h[U] \Rightarrow [m]$ s.t. $\forall x, y$, $Pr_{h \in_R H}[h(x) = h(y)] \leq \frac{1}{m}$.*

Example 5. *In $x, y \in_R 0, 1$, $x, y, x \oplus y$ are 2-wise independent.*

As a graph explanation, the professor used the graph below. For a 2-wise independent family of hash functions, it is identical to look at any pair of the columns in the table and are independent to each other. See Figure 1.

Similarly, the definition of k-wise independent is as followed [2].

Definition 6. *A k-wise independent family of hash function $\mathbb{H}$ is defined as $\forall h \in \mathbb{H}$ with $h[U] \Rightarrow [m]$ s.t. $\forall x_1, x_2, ..., x_k$, $Pr_h(h(x_1) = h(x_2) = ... = h(x_k)) = O(\frac{1}{m^k})$*

Example 7. *For k-wise independent: $h(x) := (\Sigma_{i=1}^{k} a_i x^i mod(p)) mod(m)$ s.t. $a_1, a_2, .., a_k \in^{iid} R[\mathbb{F}_p]$ and p is a prime number.*

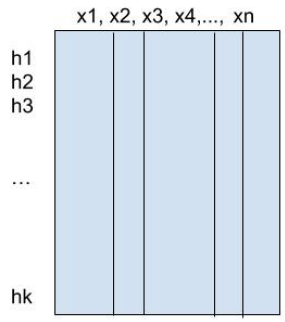

Figure 1: Each pair of column are independent

Claim 8. *The truly random is an over kill. To get* $Pr_h(h(x) = h(y)) = \frac{1}{m}$*, it is enough to take universal 2-wise independent.*

Thus, to solve the two big problems listed, we can take the two approaches as followed.

1. To get the $Pr = \frac{1}{m}$, it is enough to just take universal 2-wise independent function. More formally, $\forall x \in [U] : h(x) := ((ax+b)\, mod(P_{\geq |U|}))\, mod(m)$ by the Definition 6.

2. For the number of collisions, the current hash function has $E[collisions \in \mathbb{S}] = \Theta(\frac{n^2}{m})$. A naive solution is to set $m = 100n^2$, then the probability of a collision is $Pr_h[\exists column \in |S| < n]$. Such dictionary will have space $O(n^2)$ which is very bad, though the time is $O(1)$.

The next problem is how to get a linear space dictionary?

4 Linear Space

Previous methods have a large space requirement. To use only linear space, which means we have $m = O(n)$, we must handle collisions, for collisions will be sure to happen in this case.

4.1 Chaining

Chaining method is an implementation we often see. The first step is we use a normal hash table, with a hash function h maps our keys into corresponding address. The

difference is that when a collision happens, we use a linked list to store all the keys that are being mapped to the same slot. Below Figure 2 [1] shows a chaining hash table:

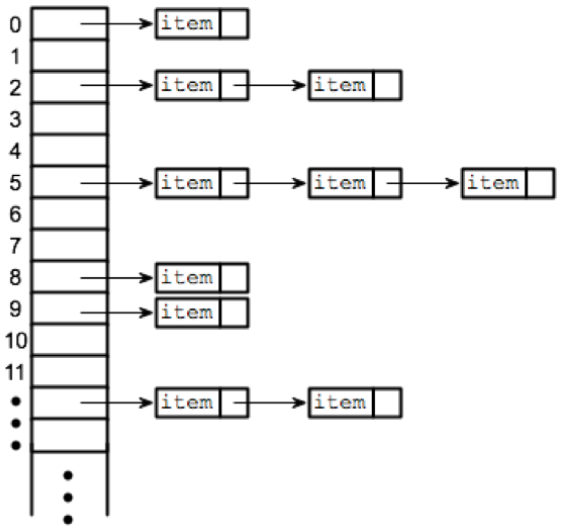

Figure 2: An example of chaining

4.1.1 Space Complexity

For a slot i, let C_i denote the length of the linked list chain in the slot i. We now calculate the expectation of this variable:

$$E[C_i] = \Sigma_j[Pr[h(x_j) = i]] = \Sigma_j[O(\frac{1}{m})] = O(\frac{n}{m})$$

Note that by choosing $m = O(n)$, the expected length will be constant, for example we can let this constant less than $\frac{1}{10}$ by choosing some proper m.

4.1.2 Time Complexity

We can prove that the $max|C_i| \leq O(\frac{lgn}{lglgn})$ by balls in bins theorem and Chernoff bound. As a result, the worst-case time complexity for a query will be $O(\frac{lgn}{lglgn})$.

Another interesting fact is that if we have a cache with $\theta(lgn)$ elements, we will have a constant amortized query time.

(Note: Professor had not provided formal proof of above claims during the class, please see more detail in section 3.1 High Probability Bounds from the reference [2])

4.2 FKS Dictionary: Perfect Hashing

FKS hashing consists of two main steps: 1. Choose a 2-wise independent hash function as first level hashing, 2. For $\forall C_i$, use hash function $h : [C_i] \to [8[C_i]^2]$. By setting up a hash table according to those two steps, we now have zero collision on the chains based on the birthday paradox and Markov.

4.2.1 Space Complexity

For 1st level hash we use a space of $O(lgn)$, which is just a cost for 2-wise independent hash function. For 2nd level hash, the space is:

$$E[space] = E[\Sigma_i C_i^2] = n + E[\Sigma_{i,j \in S} \mathbb{I}_{i,j}^2] = E[\Sigma_{i,j \in S} Pr[h(x_i) = h(x_j)]] = n + O(\frac{n^2}{m})$$

Thus, when $m = O(n)$, $E[space]$ is just $O(n)$.

4.2.2 Conclusion

We now have a dictionary with: Space $\leq 4n$ and Time $= O(1)$
What can we improve: 1. $1 + \epsilon$ n space, 2. Not parallel, 3. Can be simpler.

4.3 Alternate method to chaining

1. Linear probing
2. Cuckoo hashing
3. Tabulation
4. Bloom filter

4.3.1 Linear Probing

The idea is, given a hash function h, we try to insert x into $h(x)$, if $h(x)$ is full try $h(x) + 1$, $h(x) + 2$, and so on until a slot is empty. Below Figure 3 shows an example:

It seems linear probing is a bad idea, for "the rich get richer, the poorer get poorer". When long runs of adjacent elements develop, they are more likely to have collisions which increase their size. Due to the cache locality when the runs are not too large, linear probing is actually efficient in practice. It is only 10% slower then a normal memory read.

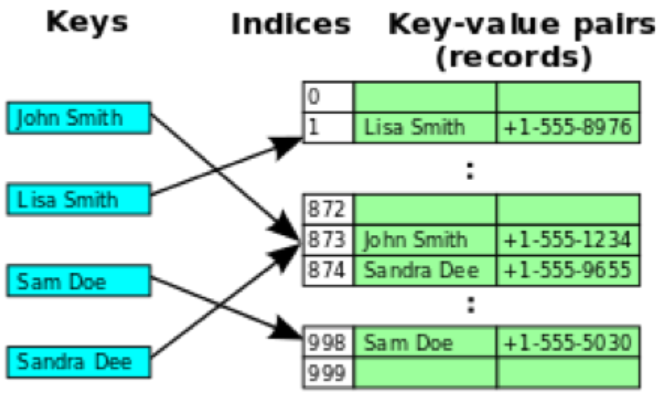

Figure 3: An example of linear probing

4.4 Next Time

Theorem: For a totally random hash function h with space $= O((1+\epsilon)n)$ the expect time complexity is $O(\frac{1}{\epsilon^2})$. For hash function which is k-independent, $k = \Omega(lgn)$ is sufficient. In [3], we saw $k = 5$ is enough.

References

[1] `https://www.hackerearth.com/practice/data-structures/hash-tables`

[2] Erik Demaine. 6.851: Advanced Data Structures, lecture 10
`https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-851-advance`

[3] Anna Pagh, Rasmus Pagh, and Milan Ruzic. Linear probing with constant independence. In In STOC '07: Proceedings of the thirty-ninth annual ACM symposium on Theory of computing, pages 318-327. ACM Press, 2007