COMS E6998 Advanced Data Structures **Spring 2019**

Lecture 11: Hashing II

Lecturer: Omri Weinstein *Scribe: Xingchen Zhou*

1 Overview

In this lecture, we look at simpler, faster (more succinct) and cache-friendly ways to deal with chaining:

(1) Linear Probing

(2) Cuckoo Hashing

(3) Tabulation Hashing

(4) Bloom Filters (didn't have time to cover this topic in this lecture)

2 Linear Probing (LP)

Linear probing handles collision by putting x at first available slot $[h(x)+i] \mod m$, where $m \geq (1+\varepsilon)n$ is required. If h(x) is full, we try $h(x)+1$, and $h(x)+2$, and so forth.

2.1 Intuition:

Intuitively, linear probing is a bad idea because "the rich get richer, and the poor get poorer"; that is to say, when long runs of adjacent elements develop, they are even more likely to result in collisions which increase their size. However, with good hash function, this won't happen.

2.2 Upshot of Linear Probing:

(1) Very simple

(2) Cache-friendly and space-efficient

(3) In practice, only 10% slower than memory access

2.3 Theorem [Knuth 1962, Pagh and Rodler 2007]:

If h is totally random, $h : [U] \to [m = (1+\varepsilon)n]$, then

$$\mathbb{E}[\text{length of run}] \leq \mathcal{O}(\frac{1}{\varepsilon^2}),$$

when inserting n keys. Here, the length of run is the query time before insertion.

Remark:

(1) This is true even for $\mathcal{O}(\lg n)$ - wise independent hash functions [Schmidt & Siegel 1990].

(2) 5 - wise independent also suffices [Pagh, Pagh, Ruzic 2007].

(3) If 4 - wise independent, then $\mathbb{E}[\text{length of run}] \geq \Omega(\lg n)$ [Patrascu & Thorup 2011].

2.4 Proof of the Theorem:

Assume a totally random hash function h over a domain of size n. To simplify the proof, we assume the size of the hash table $m = 3n$. Then, our goal is to show that

$$\mathbb{E}[\text{length of run}] \leq \mathcal{O}(1).$$

For our analysis, we will refer to an imaginary perfect binary tree T where the leaves correspond to slots in our hash table (the number of leaves is $m = 3n$). Nodes correspond to ranges of slots in the table.

For any node $v \in T$ of height $h = d(v)$, the expected number of keys which hash to a single slot is $1/3$, and thus the expected number of keys which hash to slots within a node at height h is $\frac{1}{3} \cdot 2^h$.

Now, we introduce a key definition for this proof: we call a node v of height h **dangerous** if the number of keys in the table that hash to this node is greater than $\frac{2}{3} \cdot 2^h$. Then, we apply the Chernoff bound to get

$$\begin{aligned}&\Pr[v \text{ is dangerous }]\\=&\Pr[\text{number of keys} \in T_v > 2 \cdot \mathbb{E}[\text{number of keys}]]\\\leq&(e/4)^{2^h/3}.\end{aligned}$$

Next, we relate the presence of run in tables to the existence of dangerous nodes. We consider a run of length $\in [2^l, 2^{l+1})$ for arbitrary l. We look at nodes of height $h = l - 3$ spanning this run. There are between 8 and 17 nodes (it is 17 rather than 16 because we may need an extra node to finish off the range). The first 4 nodes span more than $3 \cdot 2^h$ slots of the run, and are full with keys that must hash within the 4 nodes (via h). Furthermore, the keys occupying the slots in these nodes must have hashed within the nodes as well (they could not have landed in the left, since this would contradict our assumption that these are the first four nodes of the run).

We can prove that at least 1 of these 4 nodes is dangerous. Indeed, if all these 4 nodes are not dangerous, there would be less than $4 \cdot \frac{2}{3} \cdot 2^h = \frac{8}{3} \cdot 2^h$ keys occupied slots, which is less than the number of slots of the run we cover.

Hence, we have

$$\begin{aligned}&\Pr[\text{length of run} \in x \text{ has length } \in [2^l, 2^{l+1})]\\\leq&17 \cdot \Pr[\text{node of height } l-3 \text{ is dangerous}]\\\leq&17 \cdot (e/4)^{2^l/24}.\end{aligned}$$

Finally, to finish the proof,

$$
\begin{aligned}
&\mathbb{E}[\text{length of run}] \\
\leq &\sum_{l=1}^{\infty} 2^l \Pr[\text{length of run} \in [2^l, 2^{l+1})] \\
\leq &17 \cdot \sum_{l=1}^{\infty} 2^l \cdot (e/4)^{2^l/24} = \mathcal{O}(1).
\end{aligned}
$$

3 Cuckoo Hashing

3.1 Main Idea:

We use 2 hash tables A and B of size $m \geq (1+\varepsilon)n$, with hash functions h and g respectively. The cuckoo part of Cuckoo hashing refers to bumping out a key of one table in the event of collision, hashing it into the other table and repeating this process until the collision is resolved.

For example, to insert key x, we first try inserting x into $A[h(x)]$. If $A[h(x)]$ is free, then insert x. Otherwise, try $B[g(x)]$. If $B[g(x)]$ is occupied, put x into $B[g(x)]$, and kick the old item z in $B[g(x)]$ out to $A[h(z)]$. Continue this step recursively if needed.

If more than $6 \lg n$ keys got bumped, then rebuild the entire hash table. Our goal is to show that such Cuckoo hashing only fails with small probability.

3.2 Theorem:

Under uniform hash functions h and g,

$$\Pr[\text{a simple path in Cuckoo graph of length} > k] \leq 2^{-k}.$$

Remark: In fact, we can show that

$$\Pr[\text{Cuckoo fails/rebuilds}] \leq \frac{1}{n}.$$

It's true even for k-wise independent h and g.

3.3 Proof:

Suppose that there exists a simple path of length k, $p = x_1 x_2 \ldots x_k$ where there is no cycle, in the Cuckoo graph. If h and g: $[U] \to [m]$, where $m = 2n$, then describing h and g on $\{x_i\}_{i=1}^{n}$ requires $2n \lg m$ bits.

We will show that if there exists a simple path of length greater than k, then we can compress(encode) $h(s)$ and $g(s)$ using only $2n \lg m - k$ bits, and

$$\Pr[\text{length of path} > k] \leq 2^{-k}.$$

Finally, to finish the proof, we are left with showing the following two Claims:

Claim 1. A random variable x requires N bits to describe, Encode($x|w$) $< N - k$. The probability of this event $\Pr(w) \leq 2^{-k}$.

Quick proof of Claim 1: Let P denotes $\Pr(w)$. We can encode x with Encode($x|w$)$+ \lg \frac{1}{P}$ bits. Then Encode$(x|w) + \lg \frac{1}{P} \geq N$ implies $\lg \frac{1}{P} \geq k$. Hence, $\Pr(w) \leq 2^{-k}$.

Claim 2. Existence of compression/encoding of $h(s)$ and $g(s)$ in $2n \lg m - k$ bits:

(1) Determining path p starts in A or B takes 1 bit.

(2) Slots of nodes along path: $(k+1) \lg m$ bits.

(3) Only $\lg n$ bits per node in path p are needed to specify key $x_i \in [n]$ corresponding to edge.

4 Tabulation Hashing (Concept from 1970's)

Given a key $x \in U = [2^w]$, x is a w bits string, $x = x_1 x_2 \dots x_w$. We can chop x into a vector $\bar{x}$ of length c, $\bar{x} = (\bar{x}_1, \bar{x}_2, \dots, \bar{x}_c) \in \Sigma^c$, where $\Sigma = w/c$. Then, we create a totally random hash table T_i on each $\bar{x}_i$. This takes $\mathcal{O}(cu^{1/c})$ words of space to store and takes $\mathcal{O}(c)$ time to compute. In addition, tabulation hashing is 3-wise independent. It is defined as

$$h(x) = T_1(x_1) \oplus T_2(x_2) \oplus \dots \oplus T_c(x_c).$$

Reference

[1] Erik Demaine. 6.851: Advanced Data Structures, lecture 10

https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-851-advanced-d
calendar-and-notes/MIT6_851S12_Lec10.pdf

https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-851-advanced-d
calendar-and-notes/MIT6_851S12_L10.pdf