CS 7800/4810: Data Str & Alg Scalable Comp Spring 2026

Lecture 10 — Feb 18, 2026

Prof. Prashant Pandey *Scribe: Ilgar Mammadov*

1 Overview

In the last few lectures, we covered several topics related to hashing. These topics included:

- Basics of probability;
- Balls and Bins problem:
  - Load in the fullest bin when we have n balls and n bins is

$$O\left(\frac{\lg n}{\lg \lg n}\right) \text{ WHP} \tag{1}$$

- Hash functions (universal, k-wise independent);
- Hash tables (Chaining);

1.1 Chaining

We have n keys from universe, m buckets of hash table - where $m = \Theta(n)$, and a hash function $h(x)$.

If multiple items hash to the same bucket, then we use linked lists for chaining. Length of the longest chain is:

$$O\left(\frac{\lg n}{\lg \lg n}\right) \text{ WHP} \tag{2}$$

The expected time for insert, delete, and query operations is constant with chaining hash table. Insert operations are guaranteed to take constant time even in the worst case. However, delete and query operations are not guaranteed to take constant time in the worst case.

In today's lecture, we covered Perfect Hashing, Linear Probing, Quadratic Probing, Cuckoo Hashing and introduction to Two-choice hashing.

2 Perfect Hashing [Fredman, Komlos, Szemeredi - FKS 1984 [1]]

Goal: Queries in constant time guarantee even in the worst case.

- O(1) worst case query

- O(n) space
- O(n) expected preprocessing time

Insight: We will use two-level hashing.

Idea: Use a first-level hash table, but instead of storing chains as linked lists, store each chain in its own secondary hash table **with no collisions.**

2.1 Level 1 - Primary Hash Table

Again, the goal is to store n keys into $\Theta(n) = m$ slots. m is some multiplicative multiple of n.

$C_t = \#$ number of keys hashing to slot t.

The expected number of keys hashing to slot t is : $E[C_t] = O(1) = n/m$

2.2 Level 2 - Secondary Hash Table

For each slot t, create a hash table of size S_t, where $S_t = \Theta(C_t^2)$

Now, the question is why we choose the secondary hash table size to be C_t^2?

2.3 Analysis

If two keys, i and j are both hashed to the same chain C_t, then we will have a collision in the chain C_t if the probability that the secondary hash function of these two keys match:

$E[\#$ of collisions in table $t] = \sum_{i,j \ both \ in \ C_t} Pr[h_t(x_i) = h_t(x_j)]$

We are trying to bound the number of expected collisions in table t.

To calculate the number of possible pairs of i and j in t:

$$\binom{C_t}{2} = \frac{C_t(C_t - 1)}{2}$$

If the secondary hash function is a universal hash function and the size of the chain is S_t, then in C_t, each pair collides with probability $\leq 1/S_t$

So, we can use the number of possible pairs and the suggested secondary hash table size to calculate the expected number of collisions and prove that the chance of collision in any table is constant:

$$E[\#\text{ of collisions}] \ \leq \ \frac{C_t(C_t - 1)}{2} \cdot \frac{1}{\Theta(C_t^2)} = O(1)$$

If we set $S_t = kC_t^2$ (k is a constant):

$$E[\#\text{ of collisions}] \leq \frac{C_t^2}{2} \cdot \frac{1}{kC_t^2} = \frac{1}{2k}$$

For $k = 1$, we need 2 trials in expectation to create a hash function where we have no collision in the secondary hash tables. If we make the constant higher, probability that there will be no collision becomes higher and we do not need even 2 trials.

We did not discuss the proof of space complexity, however, it can be found in the notes by Professor.

3 Linear Probing

Linear probing is an open-addressing hash table. The differentiating factor for all open-adressing hash tables is the probing methodology.

Idea: $Insert(x)$: Put x at first available slot

$$[h(x) + i] \bmod m$$

Requirement:

$$m > (1 + \epsilon)\, n$$
$$0 < \epsilon < 1$$

For example, we hash x and find that the corresponding slot is empty, so we put x to that slot. Then we hash y, which also corresponds to the same slot, so we probe the table to find an alternative slot, which is the first empty slot.

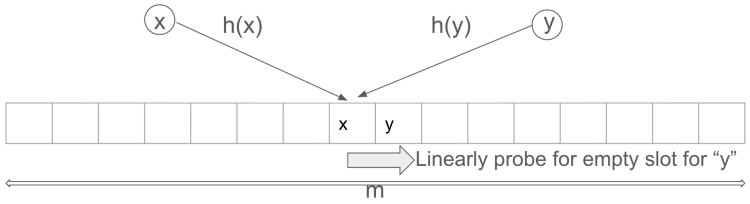

Figure 1: Linear Probing

To achieve the query operation in this hash table we first check the slot that corresponds to the hash of the key. If we found the element, then we stop. If we do not find the element in that slot, then we do linear probing and stop either if we find the element or an empty slot.

Expected cost of queries is $(1/\epsilon^2)$

4 Quadratic Probing

$$h(k, i) = \left(h(k) + i^2\right) \bmod p$$

In quadratic probing, instead of probing one slot at a time and incrementing by 1, we jump in quadratic steps, which makes cluster lengths smaller than those in linear probing and query time decreases.

Although linear probing is better for cash locality than quadratic probing, its main drawback is that as its population increases, the performance decreases.

5 Cuckoo Hashing [Pagh + Rodler 2004] [2]

2 tables with m buckets:

$$m > (1 + \epsilon)\, n$$

2 hash functions: $g \rightarrow A,\ h \rightarrow B$

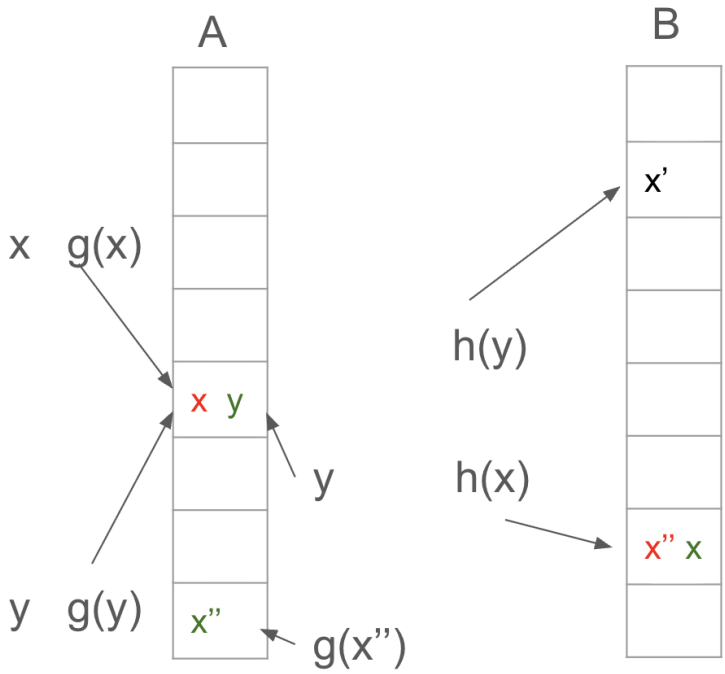

Figure 2: Cuckoo Hashing

Insertion process: First, we hash $g(x)$ and check if the slot in the table A is empty. If so, we place x there.

Then, we hash $g(y)$ and it corresponds to the same slot as x in the table A. So, we hash y using $h(y)$ and check table B for the corresponding slot. If it is also full, then we kick one of the elements in these slots randomly. Let's kick x from table A and put y into table A.

Now, we need to hash $h(x)$ and place it to table B. If the slot at table B is full, we kick that element from table B, hash it $g(x'')$ and put to the table A.

If we observe a cycle in Cuckoo hashing, then it means that the hash-tables are full.

The worst case query time and delete operation time complexities are constant $O(1)$. We just check the two slots.

Insertion time complexity is not constant.

6 Two-choice hashing

Idea: Every time pick 2 hash functions, insert to the least loaded bin.

Fullest bin will have $\lg(\lg(n))$ elements WHP in two-choice hashing.

The table has n/b buckets and each bucket will have b slots.

If the slots from both hash functions during insertion is full, then it means that we faced failure and there is no kicking in two-choice hashing. We will discuss how big b should be, but as a hint, if the number of balls is n, the number of bins b is some function of n, $f(n)$.

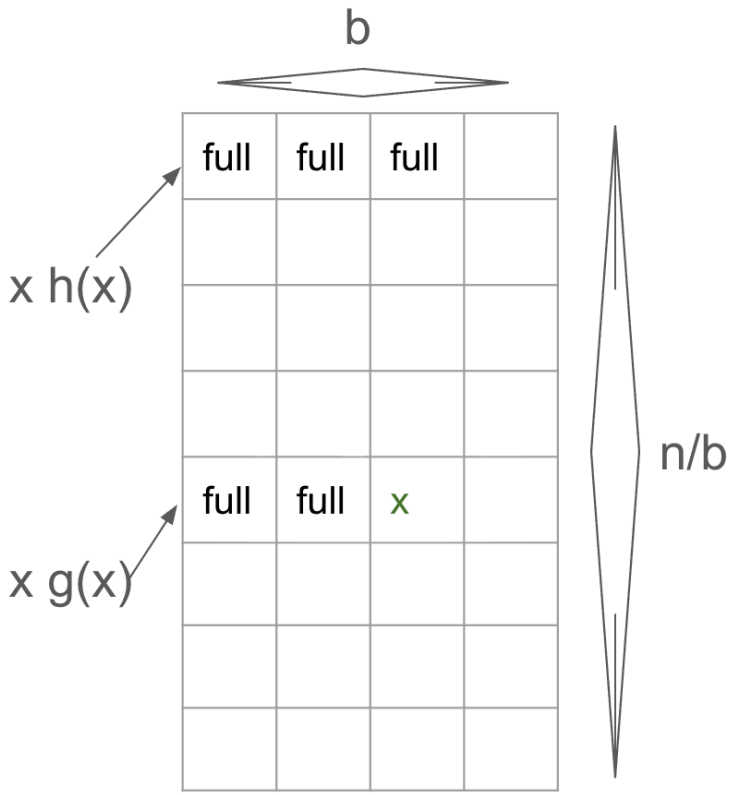

Figure 3: Two-choice Hashing

References

[1] Michael L. Fredman, János Komlós, Endre Szemerédi. Storing a Sparse Table with 0(1) Worst Case Access Time. *J. ACM*, 31(3):538-544, 1984.

[2] Rasmus Pagh, Flemming Friche Rodler. Cuckoo hashing. *J. Algorithms*, 51(2): 122-144, 2004.