12 Feb 2025

Hashing

(Typeset lecture notes coming soon!)

Balls ⇝ "keys"

Bins ⇝ "buckets"

Hashing: assign keys to buckets
randomly but reproducibly.

<u>Dictionary</u> a.k.a. Associative array, Key-value store.

Data structure that stores a set of key-value pairs $(x,v) \in X \times V$, with at most one value per key.

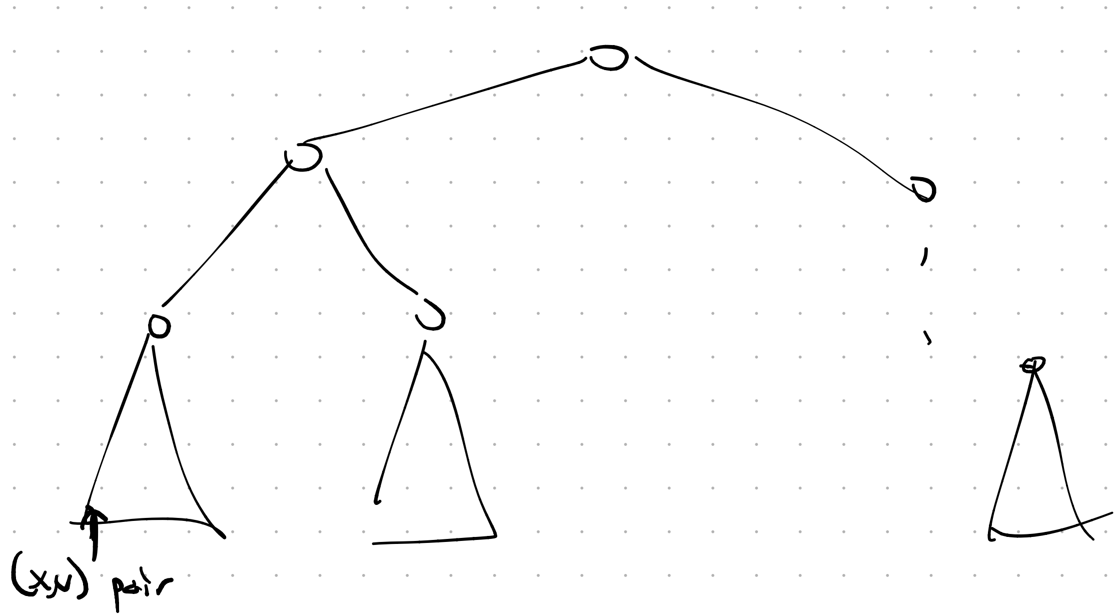
<u>Supports</u>: LOOKUP(x): returns value v stored with x
or NOT FOUND.

INSERT (x,v): insert (x,v), overwriting (x,v') if necessary

DELETE(x): remove (x,v) from data structure
if any such pair exists.

a.k.a. Python dictionaries, Java/C++ HashMaps.

<u>Deterministic implementation.</u> Balanced Binary search tree.



Tree structured s.t.

a. Every node has equal # leaves (within 1) in Left, Right subtrees.

b. All keys in left subtree are earlier in the ordering of X than the keys in the right subtree.

Assume m key-value pairs organized this way

$\Rightarrow$ tree has depth $\lceil \log_2(m) \rceil$

$\Rightarrow$ LOOKUP(x) takes $O(\log_2 m)$ time.

Space complexity: $O(m)$ tree nodes so $O(m)$ space if each key and value occupies $O(1)$ space.

More generally, the data structure occupies space $O(S)$ where S is the min amt. of storage space needed to store all keys and values.

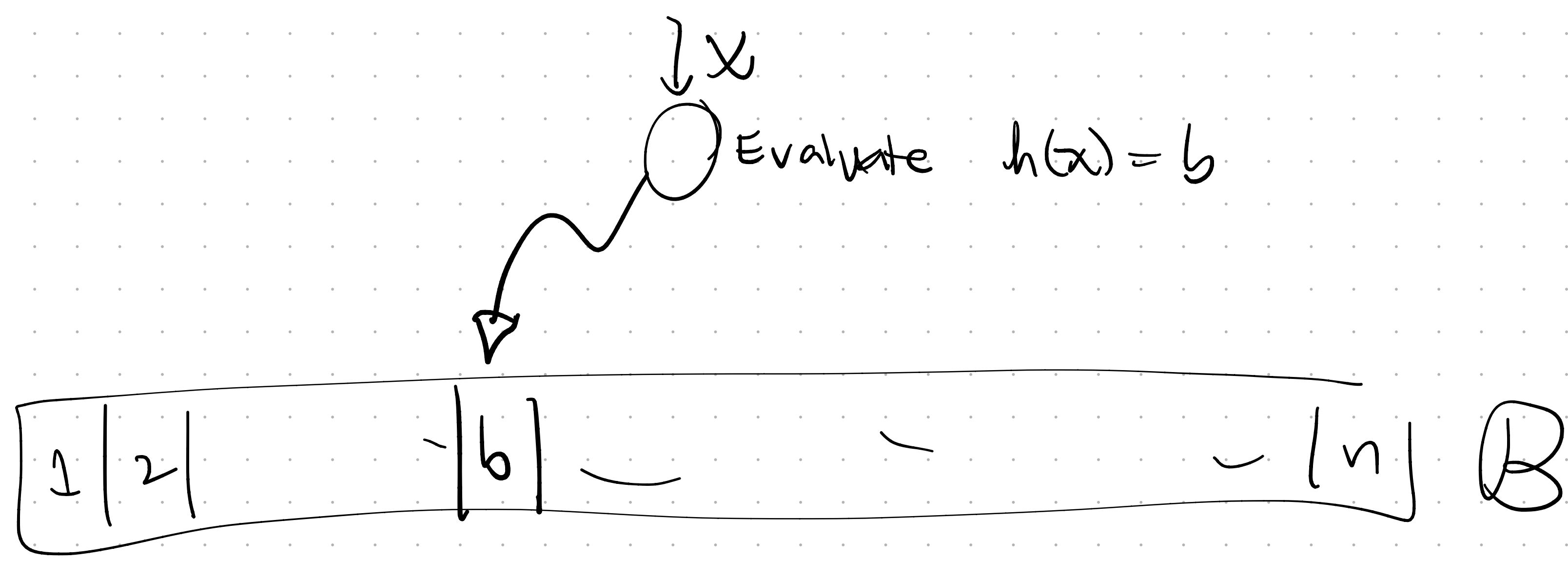
<u>Hash Table</u>: an array of n buckets. (Typically $n = O(m)$.)

Each bucket contains a linked list of key-value pairs. ("chain hashing")

A <u>hash</u> <u>function</u> maps keys (randomly or pseudorandomly) to buckets.

Let $\mathcal{B}$ denote {buckets}.

For now assume* (unrealistically) the hash function is a <u>uniformly random</u> $h: X \to \mathcal{B}$, and that evaluating $h(x)$ takes $O(1)$ time.



Lookup(x) : iterates through linked list in bucket $b = h(x)$.
If (x,v) found, return v.
Else return NOT FOUND.

INSERT(x,v): iterates through linked list b.
If (x,v') found, replace with (x,v).
Else append (x,v) to linked list

DELETE(x): iterates through linked list b.
if (x,v) found delete it
else do nothing.

Actual running time is

$$O(1 + \underbrace{\text{length of linked list stored at } h(x)}_{})$$

random variable depending on h and on the data stored in the table.

Linearity of expectation:

$$\mathbb{E}[\text{ length of linked list stored at } h(x)]$$

$$= \mathbb{E}[\# y \in \mathcal{X} \text{ s.t. a pair } (y,w) \text{ is stored in the list at } h(x)]$$

$$= \sum_{b \in \mathcal{B}} \sum_{y \in \mathcal{X}} \Pr[h(x) = h(y) = b \text{ and } (y,w) \text{ is a pair stored in the table}]$$

$\leq m$ distinct pairs (y,w) stored in table.

At most one satisfies $y = x$.

For that one $\Pr(h(x) = h(y) = b)$ is $\frac{1}{n}$.

For the other $\leq m$ pairs (y,w) with $y \neq x$

$\Pr(h(x) = h(y) = b)$ is $\frac{1}{n^2}$.

So,

$$\sum_{b \in \mathcal{B}} \sum_{y \in \mathcal{X}} \Pr[h(x) = h(y) = b \text{ and } (y,w) \text{ is a pair stored in the table}]$$

$$\leq \sum_{b \in \mathcal{B}} \left[\frac{1}{n} + \frac{m}{n^2} \right]$$

$$= 1 + \frac{m}{n} \leftarrow \text{"load factor"}$$

Typical implementations use $n \geq m$

$\Longrightarrow$ load factor ≤ 1.

Def. A pairwise independent (a.k.a. 2-universal) hash family is a probability distribution on functions $h: \mathcal{X} \to \mathcal{B}$ satisfying

$\forall\, x_1 \neq x_2$ in $\mathcal{X}$ the pair

$(h(x_1), h(x_2))$ is uniformly distributed over $\mathcal{B}^2$.

Example. Suppose $\mathcal{X} = \mathcal{B} = \{0, 1, 2, \ldots, p-1\}$

and p is prime.

Sample random h by sampling coefficients $a, b \in \{0, \ldots, p-1\}$ at random

and defining $h(x) \equiv ax + b \pmod{p}$