Probabilistic Data Structures

EECS 214

November 18, 2015

Take-aways

- What's a hash function? What makes a hash function good?
- What's the purpose of a hash table? How does it work, and how can it "go wrong"?
- What's the purpose of a Bloom filter? How does it work, and how can it "go wrong"?
- What does it mean for a data structure to be *probabilistic*?

Mappings

Remember counting byte frequencies in HW1?

Mappings

Remember counting byte frequencies in HW1?

You need(ed) a mapping from byte values to their counts:

$$\text{byteFrequencies} : \{0, 1, \ldots, 255\} \to \mathbb{N}$$

How did you represent this?

Mappings

Remember counting byte frequencies in HW1?

You need(ed) a mapping from byte values to their counts:

$$\text{byteFrequencies} : \{0, 1, \ldots, 255\} \to \mathbb{N}$$

How did you represent this?

Easy:

```
size_t byte_freqs[256];
```

Mappings

Remember counting byte frequencies in HW1?

You need(ed) a mapping from byte values to their counts:

$$\text{byteFrequencies} : \{0, 1, \ldots, 255\} \to \mathbb{N}$$

How did you represent this?

Easy:

```
size_t byte_freqs[256];
```

Arrays are *perfect* for mappings whose domain is $\{0, 1, \ldots, k\}$ for some k

Mappings

Remember counting byte frequencies in HW1?

You need(ed) a mapping from byte values to their counts:

$$\text{byteFrequencies} : \{0, 1, \ldots, 255\} \to \mathbb{N}$$

How did you represent this?

Easy:

```
size_t byte_freqs[256];
```

Arrays are *perfect* for mappings whose domain is $\{0, 1, \ldots, k\}$ for some k

Notation note: We will write $\mathbb{N}_k$ for the set $\{0, 1, \ldots, k\}$

A different domain

What if we wanted to count word frequences instead?

A different domain

What if we wanted to count word frequences instead?

We need a mapping from words (strings) to their counts:

$$\text{wordFrequences} : \{ \text{ the set of all strings } \} \to \mathbb{N}$$

How can we represent this?

A different domain

What if we wanted to count word frequences instead?

We need a mapping from words (strings) to their counts:

$$\text{wordFrequences} : \{ \text{ the set of all strings } \} \to \mathbb{N}$$

How can we represent this?

We can't use strings to index into an array—we need a *hash function*

Definition: *hash function*

A *hash function* for some type maps values of that type to $\mathbb{N}_k$

Definition: *hash function*

A *hash function* for some type maps values of that type to $\mathbb{N}_k$

Here is a **really bad** hash function for strings:

$$hash_k(c_1 \ldots c_n) = \sum_{i=1}^{n} ord(c_i) \mod k$$

Definition: *hash function*

A *hash function* for some type maps values of that type to $\mathbb{N}_k$

Here is a **really bad** hash function for strings:

$$hash_k(c_1 \ldots c_n) = \sum_{i=1}^{n} ord(c_i) \mod k$$

It adds up the character values

How do we use a hash function?

A *hash function* for some type maps values of that type to $\mathbb{N}_k$

How do we use a hash function?

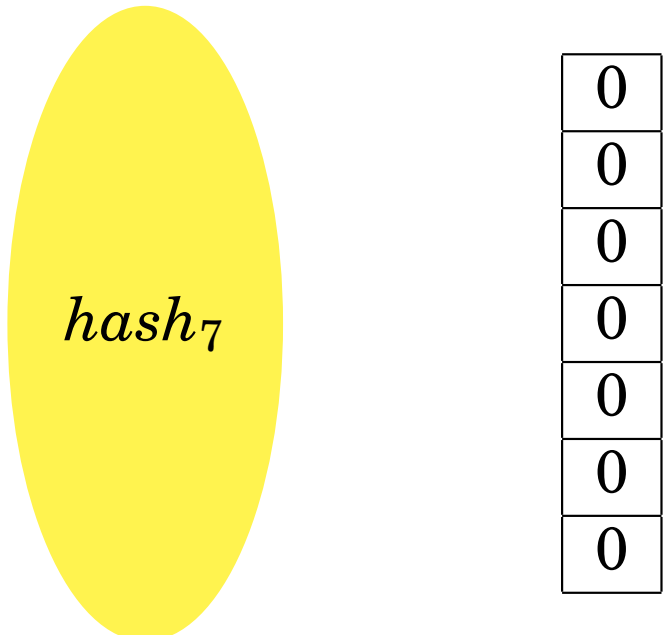
A *hash function* for some type maps values of that type to indices into an array of size k

How do we use a hash function?

A *hash function* for some type maps values of that type to $\mathbb{N}_k$
We then store our value v at the index given by $hash(v)$.

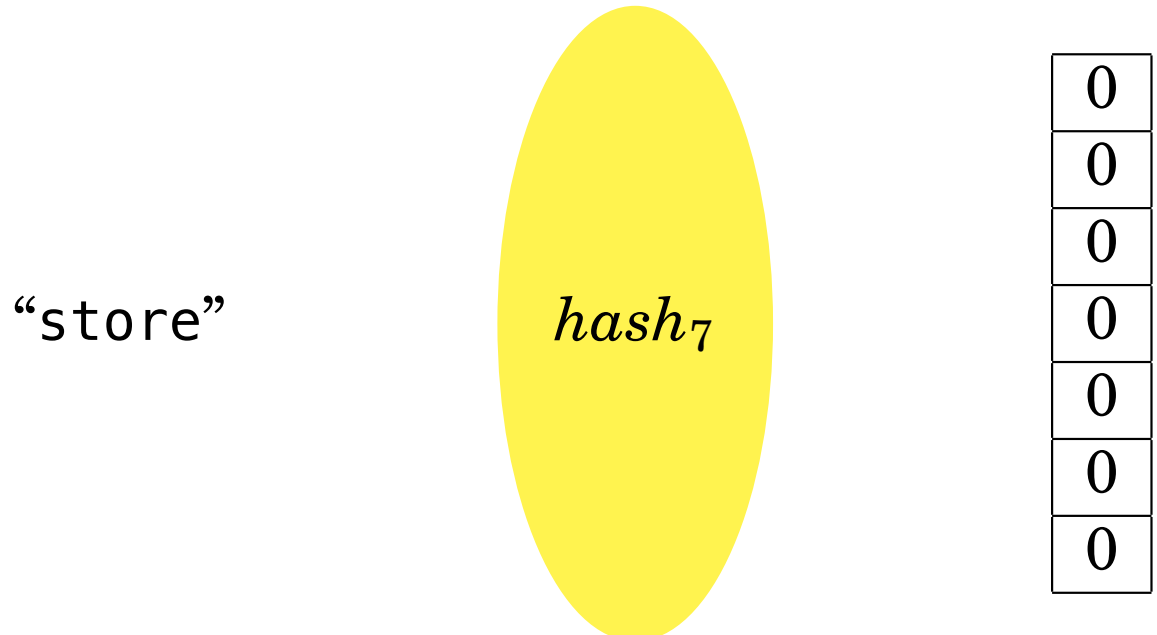
Example (using shitty string hash function)

Store word frequencies at the index given by the hash of the word:



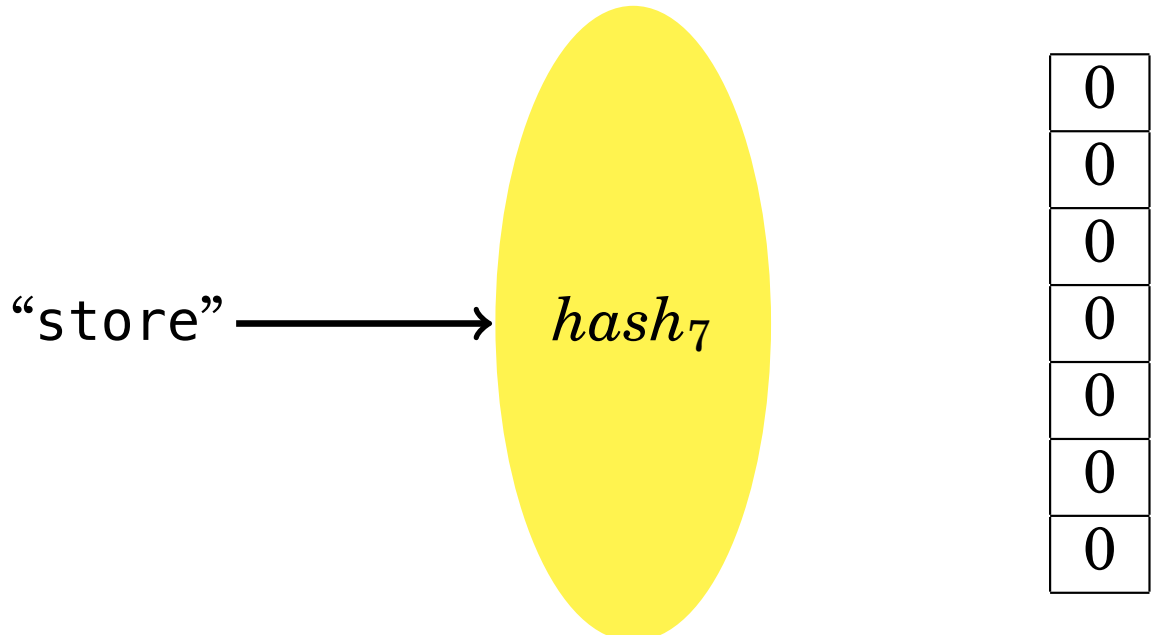
Example (using shitty string hash function)

Store word frequencies at the index given by the hash of the word:



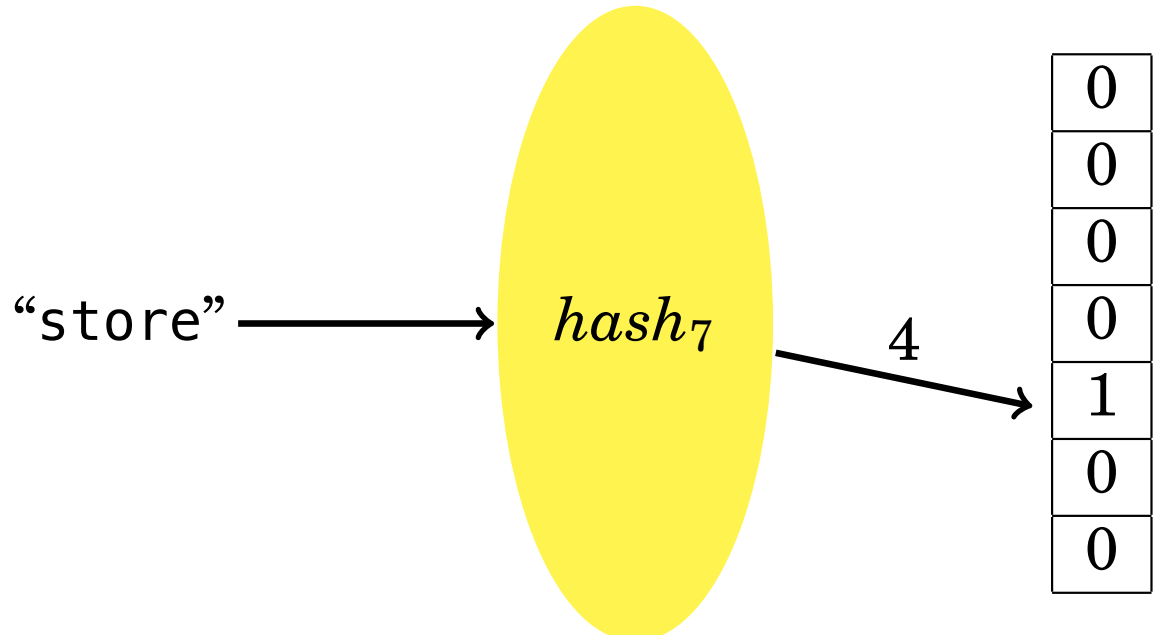
Example (using shitty string hash function)

Store word frequencies at the index given by the hash of the word:



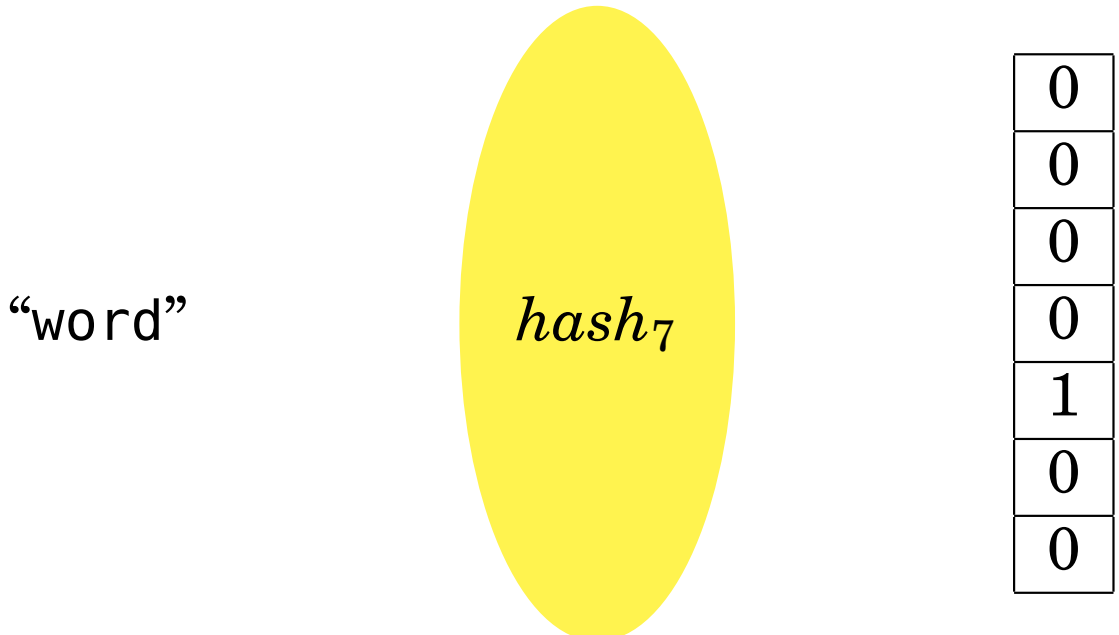
Example (using shitty string hash function)

Store word frequencies at the index given by the hash of the word:



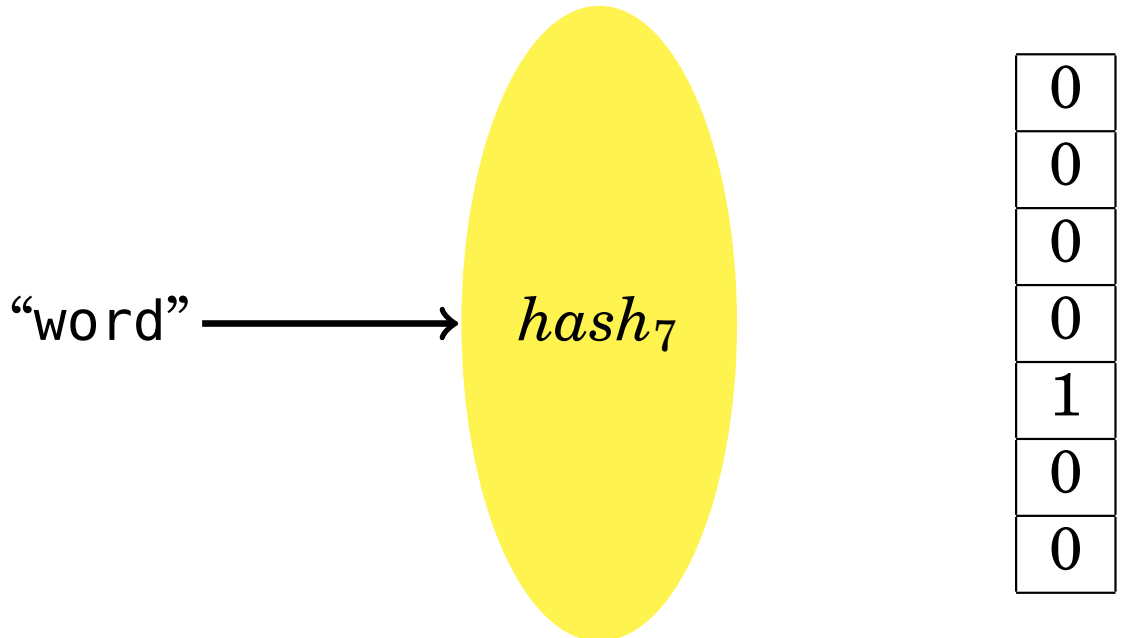
Example (using shitty string hash function)

Store word frequencies at the index given by the hash of the word:



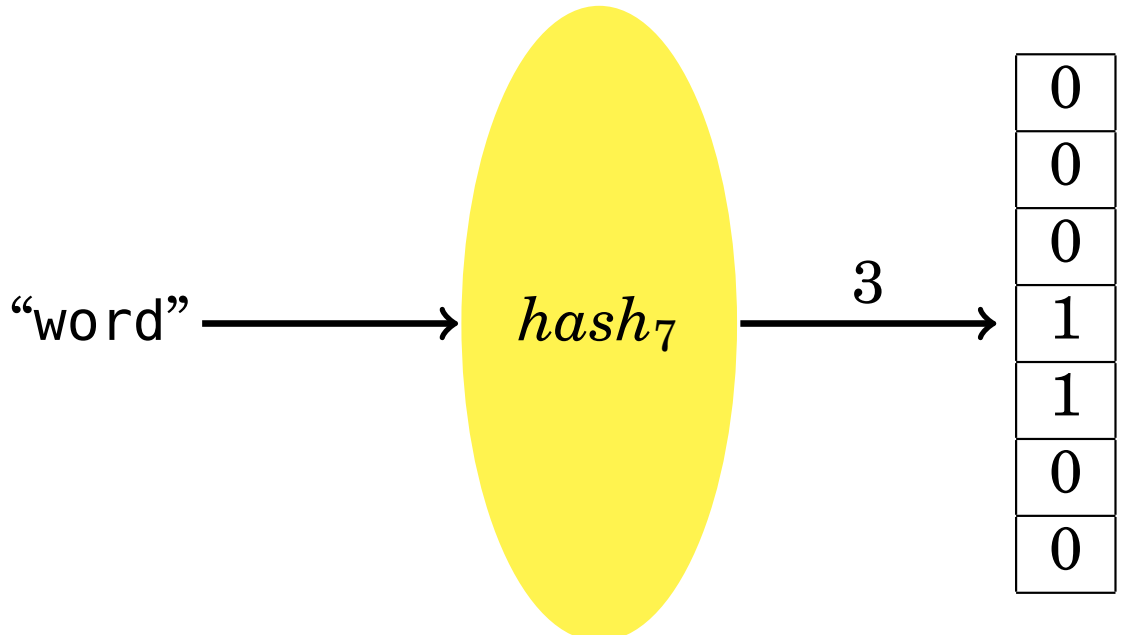
Example (using shitty string hash function)

Store word frequencies at the index given by the hash of the word:



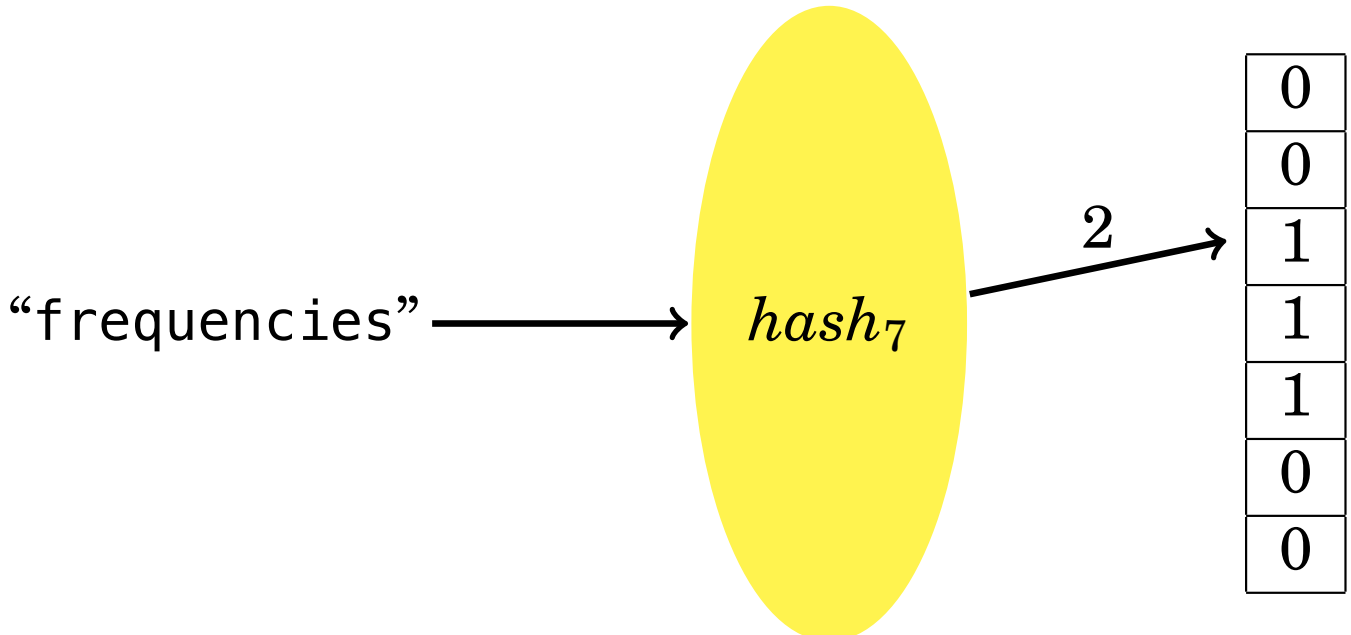
Example (using shitty string hash function)

Store word frequencies at the index given by the hash of the word:



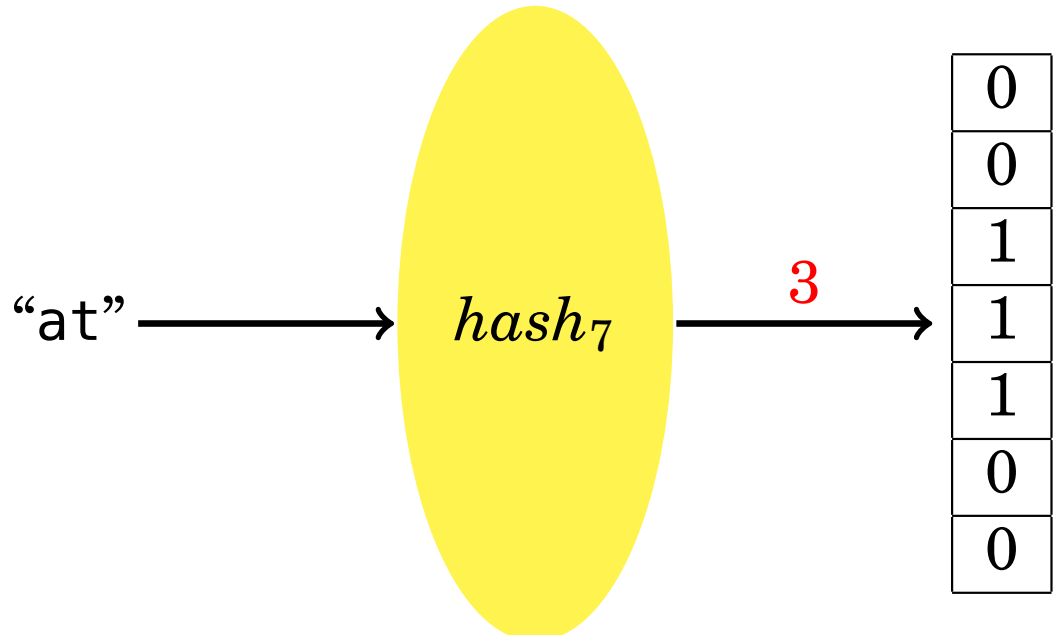
Example (using shitty string hash function)

Store word frequencies at the index given by the hash of the word:



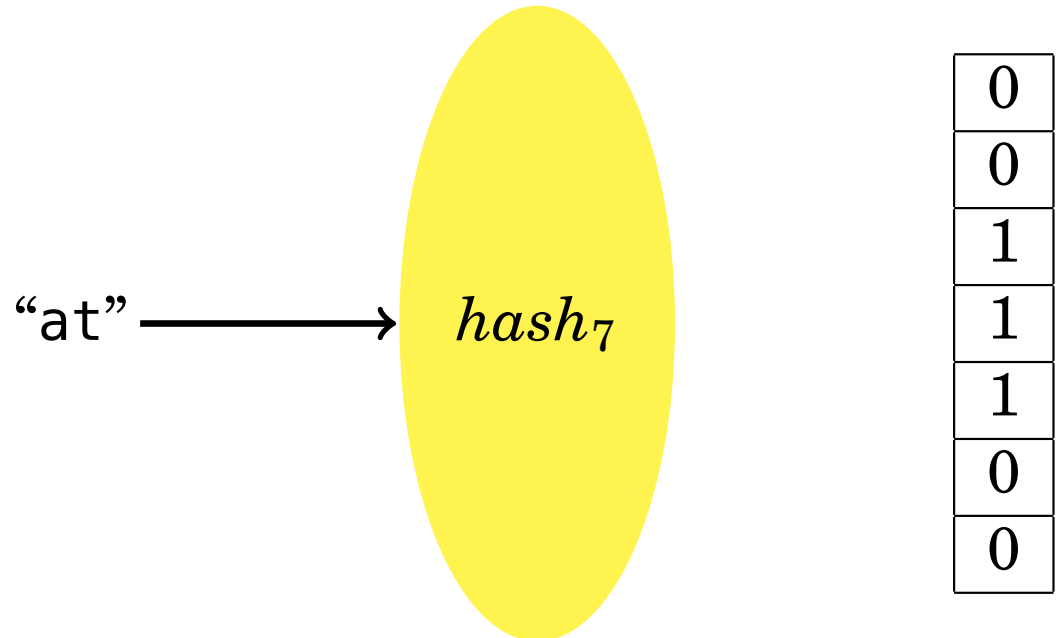
Example (using shitty string hash function)

Store word frequencies at the index given by the hash of the word:



Example (using shitty string hash function)

Store word frequencies at the index given by the hash of the word:



Hash collision!

Take two: separate chaining

Each bucket stores a linked list of associations:

(hash, 1)
(frequencies, 1) $\rightarrow$ (by, 1)
(word, 2) $\rightarrow$ (at, 1) $\rightarrow$ (of, 1)
(store, 1) $\rightarrow$ (index, 1)
(given, 1)
(the, 3)

Take two: separate chaining

Each bucket stores a linked list of associations:

(hash, 1)
(frequencies, 1) $\rightarrow$ (by, 1)
(word, 2) $\rightarrow$ (at, 1) $\rightarrow$ (of, 1)
(store, 1) $\rightarrow$ (index, 1)
(given, 1)
(the, 3)

Time complexity of hash table operations

What's the time complexity of insert? Lookup?

Time complexity of hash table operations

What's the time complexity of insert? Lookup?

Depends on how many collisions we have!

Time complexity of hash table operations

What's the time complexity of insert? Lookup?

Depends on how many collisions we have!

If we avoid collisions: $\mathcal{O}(1)$ *on average*

Time complexity of hash table operations

What's the time complexity of insert? Lookup?

Depends on how many collisions we have!

If we avoid collisions: $\mathcal{O}(1)$ *on average*

But too many collisions and the lists get too long: $\mathcal{O}(n)$

Probabilities of collisions

Number of **32-bit** hash values	Number of **64-bit** hash values	Number of **160-bit** hash values	Odds of a hash collision
77163	5.06 billion	1.42×10^{24}	1 in 2
30084	1.97 billion	5.55×10^{23}	1 in 10
9292	609 million	1.71×10^{23}	1 in 100
2932	192 million	5.41×10^{22}	1 in 1000
927	60.7 million	1.71×10^{22}	1 in 10000
294	19.2 million	5.41×10^{21}	1 in 100000
93	6.07 million	1.71×10^{21}	1 in a million
30	1.92 million	5.41×10^{20}	1 in 10 million
10	607401	1.71×10^{20}	1 in 100 million
	192077	5.41×10^{19}	1 in a billion
	60740	1.71×10^{19}	1 in 10 billion
	19208	5.41×10^{18}	1 in 100 billion
	6074	1.71×10^{18}	1 in a trillion
	1921	5.41×10^{17}	1 in 10 trillion
	608	1.71×10^{17}	1 in 100 trillion
	193	5.41×10^{16}	1 in 10^{15}
	61	1.71×10^{16}	1 in 10^{16}
	20	5.41×10^{15}	1 in 10^{17}
	7	1.71×10^{15}	1 in 10^{18}

Odds of a full house in poker
1 in 693

Odds of four-of-a-kind in poker
1 in 4164

Odds of being struck by lightning
1 in 576000

Odds of winning a 6/49 lottery
1 in 13.9 million

Odds of dying in a shark attack
1 in 300 million

Odds of a meteor landing on your house
1 in 182 trillion

10:31

What makes a good hash function?

Inputs get scattered all over the range of the output

What makes a good hash function?

Inputs get scattered all over the range of the output

Stronger: changing any one bit of the input changes each bit of the output with probability $\frac{1}{2}$

Take-aways

- What's a hash function? What makes a hash function good?
- What's the purpose of a hash table? How does it work, and how can it "go wrong"?
- What's the purpose of a Bloom filter? How does it work, and how can it "go wrong"?
- What does it mean for a data structure to be *probabilistic*?