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?

\[ \text{byteFrequencies} : 0, 1, \ldots, 255 \]

How did you represent this?

Easy:

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

Arrays are perfect for mappings whose domain is \(N\) for some \(k\).

Notation note: We will write \(N^k\) for the set \(\{0, 1, \ldots, k\}\).
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?
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:

```
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:

\[
\text{byteFrequencies} : \{0, 1, \ldots, 255\} \rightarrow \mathbb{N}
\]

How did you represent this?
Easy:

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

Arrays are \textit{perfect} for mappings whose domain is \{0, 1, \ldots, k\} for some \(k\)

\textit{Notation note: We will write }\mathbb{N}_k\text{ for the set }\{0, 1, \ldots, k\}\n
3:5
A different domain

What if we wanted to count word frequencies instead?
A different domain

What if we wanted to count word frequencies instead?

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

wordFrequencies : \{ \text{the set of all strings} \} \rightarrow \mathbb{N}

How can we represent this?
A different domain

What if we wanted to count word frequencies instead?
We need a mapping from words (strings) to their counts:

\[
\text{wordFrequencies} : \{ \text{the set of all strings } \} \rightarrow \mathbb{N}
\]

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:

$$\text{hash}_k(c_1 \ldots c_n) = \sum_{i=1}^{n} \text{ord}(c_i) \mod k$$
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:

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

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:

\[
\text{hash}_7
\]
Example (using shitty string hash function)

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

```
<table>
<thead>
<tr>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>
```

“store”
Example (using shitty string hash function)

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

```
<table>
<thead>
<tr>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>
```

“store” $\rightarrow$ hash$_7$
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:

```
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
```

```
    "word"

  hash_7
```
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:

```
<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```

Hash collision!

"frequencies" → $hash_7$ → 2
Example (using shitty string hash function)

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

```
hash
  7
  0
  0
  1
  1
  0
  0
```

Hash collision!

```
3
0
0
1
1
0
0
```

"at"
Example (using shitty string hash function)

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

"at" $\rightarrow$ hash$_7$

Hash collision!
Take two: separate chaining

Each bucket stores a linked list of associations:

<table>
<thead>
<tr>
<th>(hash, 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(frequencies, 1) → (by, 1)</td>
</tr>
<tr>
<td>(word, 2) → (at, 1) → (of, 1)</td>
</tr>
<tr>
<td>(store, 1) → (index, 1)</td>
</tr>
<tr>
<td>(given, 1)</td>
</tr>
<tr>
<td>(the, 3)</td>
</tr>
</tbody>
</table>
Take two: separate chaining

Each bucket stores a linked list of associations:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(hash, 1)</td>
<td></td>
</tr>
<tr>
<td>(frequencies, 1) → (by, 1)</td>
<td></td>
</tr>
<tr>
<td>(word, 2) → (at, 1) → (of, 1)</td>
<td></td>
</tr>
<tr>
<td>(store, 1) → (index, 1)</td>
<td></td>
</tr>
<tr>
<td>(given, 1)</td>
<td></td>
</tr>
<tr>
<td>(the, 3)</td>
<td></td>
</tr>
</tbody>
</table>
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: $O(1)$ 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: $O(1)$ *on average*

But too many collisions and the lists get too long: $O(n)$
### Probabilities of collisions

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

- 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
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 \textit{probabilistic}?