Late class start: Enjoy the solar eclipse!

Class begins at 2:15 PM.
Algorithms and Data Structures for Data Science

Hashing

CS 277
Brad Solomon

April 8, 2024

Department of Computer Science
Learning Objectives

Motivate and define a hash table

Discuss what a ‘good’ hash function looks like

Identify a key weakness of the hash table

Introduce strategies to ‘correct’ this weakness
Data Structure Review

I have a collection of books and I want to store them in a dictionary!

What data structures can I use here?

- Key ➔ Value

- Book contents
  - 2 lists: keys & values
  - match their indexing

- Tree (Binary Search Tree)
If we recognize that libraries are ordered: $O(\log n)$
What if $O(\log n)$ isn’t good enough?
A Hash Table based Dictionary

ISBN: 9781526602381
Call #: PR 6068.O93 H35 1998

ISBN: 9781526602381
Call #: PR 6068.O93 H35 1998

Chapter 1
AN UNEXPECTED PARTY

ISBN: 9781526602381
Call #: PR 6068.O93 H35 1998

ISBN: 9781526602381
Call #: PR 6068.O93 H35 1998

ISBN: 9781526602381
Call #: PR 6068.O93 H35 1998

ISBN: 9781526602381
Call #: PR 6068.O93 H35 1998

ISBN: 9781526602381
Call #: PR 6068.O93 H35 1998

ISBN: 9781526602381
Call #: PR 6068.O93 H35 1998

ISBN: 9781526602381
Call #: PR 6068.O93 H35 1998

ISBN: 9781526602381
Call #: PR 6068.O93 H35 1998

ISBN: 9781526602381
Call #: PR 6068.O93 H35 1998

ISBN: 9781526602381
Call #: PR 6068.O93 H35 1998

ISBN: 9781526602381
Call #: PR 6068.O93 H35 1998

ISBN: 9781526602381
Call #: PR 6068.O93 H35 1998

ISBN: 9781526602381
Call #: PR 6068.O93 H35 1998
A Hash Table based Dictionary

```python
1 d = {}
2 d[k] = v
3 print(d[k])
```

**A Hash Table** consists of three things:

1. **A hash function**  
   
   \[ \text{key} \rightarrow \text{int} \]

2. **A list**  
   
   \( \text{stores our data \& int} \)

3. ???
Hash Function

Maps a **keyspace**, a (mathematical) description of the keys for a set of data, to a set of integers.
A hash function **must** be:

- **Deterministic**: Given the same key twice, return the same value.
- **Efficient**: $O(1)$
- Defined for a certain size table:
  $$\text{Universe} \rightarrow 0, \ldots, M-1$$
  $M$ unique values
Hash Function

(Agrave, CS 241)
(Beckman, CS 421)
(Challon, CS 125)
(Davis, CS 101)
(Evans, CS 225)
(Fagen-Ulmschneider, CS 107)
(Gunter, CS 422)
(Herman, CS 233)

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agrave</td>
<td>241</td>
</tr>
<tr>
<td>Beckman</td>
<td>421</td>
</tr>
<tr>
<td>Challon</td>
<td>125</td>
</tr>
</tbody>
</table>

Hash function $K[0] - 'A'$

$A = 6$
$B = 1$
$C = 2$

...
Hash Function

I to 1 mapping of specific names to this specific array

(Angrave, CS 241)
(Beckman, CS 421)
(Challon, CS 125)
(Davis, CS 101)
(Evans, CS 225)
(Fagen-Ulmschneider, CS 107)
(Gunter, CS 422)
(Herman, CS 233)

Hash function

(key[0] - 'A')
General Hash Function

An $O(1)$ deterministic operation that maps all keys in a universe $U$ to a defined range of integers $[0,\ldots,m-1]$

• A hash: Function that converts any $k$ in universe to a $H$
  ($H$ could be any number)
  $\text{hash } \equiv g_0 \mod m$

• A compression: Takes our $H$ and converts to $[0,\ldots,m-1]$

Choosing a good hash function is tricky...
• Don’t create your own!
Hash Function

\[ h(k) = (k \cdot firstName[0] + k \cdot lastName[0]) \mod m \]

- Author 1: Is it hash function?
  - Not a hash function
  - Not deterministic

- Author 2: Efficient
  - Deterministic

- Is it a hash?
  - Yes

- Is universe of all books fine?
  - No

- Author 2: Not a hash function

- Author 2: Not deterministic

\[ h(k) = (\text{rand()} \cdot k \cdot \text{numPages}) \mod m \]

- Author 2: Not a hash function

- Author 2: Not deterministic

\[ h(k) = (\text{Order I insert} \ [\text{Order seen}]) \mod m \]

- Author 2: Not a hash function

- Author 2: Not deterministic
Hash Function

Exercise for viewer

Let's come up with a hash function for books.
A Hash Table based Dictionary

Author Name
Hash Function

'J' + 'T' = 30

The Hobbit

... 27 ∅ 28 ∅ 29 ∅ 30 The Hobbit 31 ∅ ...

...
A Hash Table based Dictionary

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>The Hobbit</td>
<td></td>
</tr>
</tbody>
</table>

The Hobbit

J.R.R. TOLKIEN

Chapter 1
AN UNEXPECTED PARTY

The Hobbit was a very soft-to-the-touch, and his name...
A Hash Table based Dictionary

Author Name Hash Function

‘R’ + ‘S’ = 37

<table>
<thead>
<tr>
<th>Author Name</th>
<th>Hash Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goosebumps</td>
<td>37</td>
</tr>
<tr>
<td>The Hobbit</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>31</td>
</tr>
<tr>
<td></td>
<td>38</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
A Hash Table based Dictionary

Aardvarks
Anonymous
By Jim Truth

Author Name
Hash Function

‘J’ + ‘T’ = 30

30
31
...
37
38
...

The Hobbit
∅
∅
Goosebumps
∅
Hash Collision

A **hash collision** occurs when multiple unique keys hash to the same value.

J.R.R Tolkien = 30!

Jim Truth = 30!

---

<table>
<thead>
<tr>
<th>J.R.R Tolkien</th>
<th>30</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jim Truth</td>
<td>30</td>
</tr>
<tr>
<td>Aardvarks</td>
<td>Anonymous</td>
</tr>
<tr>
<td>J.R.R Tolkien</td>
<td>Goosebumps</td>
</tr>
<tr>
<td>37</td>
<td>38</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Perfect Hashing

If $m \geq S$, we can write a perfect hash with no collisions.

$m$ elements

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
General Purpose Hashing

In CS 277, we want our hash functions to work in general.
General Purpose Hashing

If \( m < U \), there must be at least one hash collision.
General Purpose Hashing

By fixing $h$, we open ourselves up to adversarial attacks.
A Hash Table based Dictionary

User Code (is a map):

```
1 Dictionary<KeyType, ValueType> d;
2 d[k] = v;
```

A **Hash Table** consists of three things:

1. A hash function

2. A data storage structure

3. A method of addressing *hash collisions*
Open vs Closed Hashing

Addressing hash collisions depends on your storage structure.

- **Open Hashing:** Stores key, value externally
  - Linked list
  - Con store individual nodes anywhere

- **Closed Hashing:** Stores k, v internally
  - Allocate fixed memory
Open Hashing

In an open hashing scheme, key-value pairs are stored externally (for example as a linked list).
A hash collision in an open hashing scheme can be resolved by adding to the linked list. This is called separate chaining.
### Insertion (Separate Chaining)

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Hash</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>B+</td>
<td>2</td>
</tr>
<tr>
<td>Anna</td>
<td>A-</td>
<td>4</td>
</tr>
<tr>
<td>Alice</td>
<td>A+</td>
<td>4</td>
</tr>
<tr>
<td>Betty</td>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>Brett</td>
<td>A-</td>
<td>2</td>
</tr>
<tr>
<td>Greg</td>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>Sue</td>
<td>B</td>
<td>7</td>
</tr>
<tr>
<td>Ali</td>
<td>B+</td>
<td>4</td>
</tr>
<tr>
<td>Laura</td>
<td>A</td>
<td>7</td>
</tr>
<tr>
<td>Lily</td>
<td>B+</td>
<td>7</td>
</tr>
</tbody>
</table>

- `insert("Bob")`
- `insert("Anna")`

```
  0  ∅
  1  ∅  ← Bob
  2  ∅  ← Anna
  3  ∅
  4  ∅
  5  ∅
  6  ∅
  7  ∅
  8  ∅
  9  ∅
 10  ∅
```

[DNF explanation]

- The table shows the insertion of keys into a hash table using separate chaining.
- The hash values are calculated for each entry.
- After inserting "Bob" and "Anna", the hash table is shown with corresponding entries.
Insertion (Separate Chaining)  \_insert(“Alice”)

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Hash</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>B+</td>
<td>2</td>
</tr>
<tr>
<td>Anna</td>
<td>A-</td>
<td>4</td>
</tr>
<tr>
<td><strong>Alice</strong></td>
<td><strong>A+</strong></td>
<td><strong>4</strong></td>
</tr>
<tr>
<td>Betty</td>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>Brett</td>
<td>A-</td>
<td>2</td>
</tr>
<tr>
<td>Greg</td>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>Sue</td>
<td>B</td>
<td>7</td>
</tr>
<tr>
<td>Ali</td>
<td>B+</td>
<td>4</td>
</tr>
<tr>
<td>Laura</td>
<td>A</td>
<td>7</td>
</tr>
<tr>
<td>Lily</td>
<td>B+</td>
<td>7</td>
</tr>
</tbody>
</table>

```
_insert(“Alice”)
```
Insertion (Separate Chaining)

Where does Alice end up relative to Anna in the chain?

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Hash</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>B+</td>
<td>2</td>
</tr>
<tr>
<td>Anna</td>
<td>A-</td>
<td>4</td>
</tr>
<tr>
<td>Alice</td>
<td>A+</td>
<td>4</td>
</tr>
<tr>
<td>Betty</td>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>Brett</td>
<td>A-</td>
<td>2</td>
</tr>
<tr>
<td>Greg</td>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>Sue</td>
<td>B</td>
<td>7</td>
</tr>
<tr>
<td>Ali</td>
<td>B+</td>
<td>4</td>
</tr>
<tr>
<td>Laura</td>
<td>A</td>
<td>7</td>
</tr>
<tr>
<td>Lily</td>
<td>B+</td>
<td>7</td>
</tr>
</tbody>
</table>
Insertion (Separate Chaining)

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Hash</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>B+</td>
<td>2</td>
</tr>
<tr>
<td>Anna</td>
<td>A-</td>
<td>4</td>
</tr>
<tr>
<td>Alice</td>
<td>A+</td>
<td>4</td>
</tr>
<tr>
<td>Betty</td>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>Brett</td>
<td>A-</td>
<td>2</td>
</tr>
<tr>
<td>Greg</td>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>Sue</td>
<td>B</td>
<td>7</td>
</tr>
<tr>
<td>Ali</td>
<td>B+</td>
<td>4</td>
</tr>
<tr>
<td>Laura</td>
<td>A</td>
<td>7</td>
</tr>
<tr>
<td>Lily</td>
<td>B+</td>
<td>7</td>
</tr>
</tbody>
</table>

Insertion (Separate Chaining)
**Insertion (Separate Chaining)**

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Hash</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>B+</td>
<td>2</td>
</tr>
<tr>
<td>Anna</td>
<td>A-</td>
<td>4</td>
</tr>
<tr>
<td>Alice</td>
<td>A+</td>
<td>4</td>
</tr>
<tr>
<td>Betty</td>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>Brett</td>
<td>A-</td>
<td>2</td>
</tr>
<tr>
<td>Greg</td>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>Sue</td>
<td>B</td>
<td>7</td>
</tr>
<tr>
<td>Ali</td>
<td>B+</td>
<td>4</td>
</tr>
<tr>
<td>Laura</td>
<td>A</td>
<td>7</td>
</tr>
<tr>
<td>Lily</td>
<td>B+</td>
<td>7</td>
</tr>
</tbody>
</table>

0: \emptyset  
1: \emptyset  
2: B  
3: \emptyset  
4: B  
5: \emptyset  
6: \emptyset  
7: \emptyset  
8: \emptyset  
9: \emptyset  
10: \emptyset  

- **Bob**: B+ at index 2
- **Anna**: A- at index 4
- **Alice**: A+ at index 4
- **Betty**: B at index 2
- **Brett**: A- at index 2
- **Greg**: A at index 0
- **Sue**: B at index 7
- **Ali**: B+ at index 4
- **Laura**: A at index 7
- **Lily**: B+ at index 7

**Insertion Diagram**: Illustrates the placement of keys and values in the separate chaining method.
## Insertion (Separate Chaining)

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Hash</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>B+</td>
<td>2</td>
</tr>
<tr>
<td>Anna</td>
<td>A-</td>
<td>4</td>
</tr>
<tr>
<td>Alice</td>
<td>A+</td>
<td>4</td>
</tr>
<tr>
<td>Betty</td>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>Brett</td>
<td>A-</td>
<td>2</td>
</tr>
<tr>
<td>Greg</td>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>Sue</td>
<td>B</td>
<td>7</td>
</tr>
<tr>
<td>Ali</td>
<td>B+</td>
<td>4</td>
</tr>
<tr>
<td>Laura</td>
<td>A</td>
<td>7</td>
</tr>
<tr>
<td>Lily</td>
<td>B+</td>
<td>7</td>
</tr>
</tbody>
</table>

**Diagram:**

```
1  2  3  4  5  6  7  8  9  10
∅  B  A- ∅  ∅  ∅  ∅  ∅  ∅  ∅
```

- Greg: A, ∅
- Brett: A-, ∅
- Betty: B, ∅
- Bob: B+, ∅
- Ali: B+, ∅
- Alice: A+, ∅
- Anna: A-, ∅
- Sue: B, ∅
- Laura: A, ∅
Find (Separate Chaining)

- \_find(“Sue”) \(\Theta O(n)\)

<table>
<thead>
<tr>
<th>Key</th>
<th>Hash</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sue</td>
<td>7</td>
</tr>
</tbody>
</table>
Remove (Separate Chaining)

Remove ("Betty")

\[ \text{\_remove("Betty")} \]

\[ \subseteq O(n) \]

1. Find Betty \[ O(n) \]
2. Remove Betty \[ O(1) \]
Hash Table (Separate Chaining)

For hash table of size $m$ and $n$ elements:

Find runs in: $O(n)$

Insert runs in: $O(1)$

Remove runs in: $O(n)$
Fundamentals of Probability

Imagine you roll a pair of six-sided dice.

The **sample space** $\Omega$ is the set of all possible outcomes.

An **event** $E \subseteq \Omega$ is any subset.
Imagine you roll a pair of six-sided dice. What is the expected value?

The **expectation** of a (discrete) random variable is:

\[
E[X] = \sum_{x \in \Omega} (Pr\{X = x\} \cdot x)
\]

Average expected outcome

\[
\frac{1}{6} \cdot 1 + \frac{1}{6} \cdot 2 + \frac{1}{6} \cdot 3 + \ldots = 3.5
\]
Fundamentals of Probability

Imagine you roll a pair of six-sided dice. What is the expected value?

**Linearity of Expectation:** For any two random variables $X$ and $Y$, 

$$E[X + Y] = E[X] + E[Y]$$
Imagine you roll a pair of six-sided dice. What is the expected value?

**Linearity of Expectation:** For any two random variables $X$ and $Y$,

$$E[X + Y] = E[X] + E[Y]$$

$$= \sum_{x} \sum_{y} \Pr\{X = x, Y = y\} (x + y)$$
Fundamentals of Probability

Imagine you roll a pair of six-sided dice. What is the expected value?

**Linearity of Expectation:** For any two random variables $X$ and $Y$,

$$E[X + Y] = E[X] + E[Y]$$

$$= \sum x \sum y Pr\{X = x, Y = y\} (x + y)$$

$$= \sum x \sum y Pr\{X = x, Y = y\} + \sum y \sum x Pr\{X = x, Y = y\}$$

Sums up to 1
Fundamentals of Probability

Imagine you roll a pair of six-sided dice. What is the expected value?

**Linearity of Expectation:** For any two random variables $X$ and $Y$,

$$E[X + Y] = E[X] + E[Y]$$

$$= \sum_x \sum_y Pr\{X = x, Y = y\}(x + y)$$

$$= \sum_x x \sum_y Pr\{X = x, Y = y\} + \sum_y y \sum_x Pr\{X = x, Y = y\}$$

$$= \sum_x x \cdot Pr\{X = x\} + \sum_y y \cdot Pr\{Y = y\}$$
Imagine you roll a pair of six-sided dice. What is the expected value?

**Linearity of Expectation:** For any two random variables $X$ and $Y$,

$$E[X + Y] = E[X] + E[Y]$$

$$3.5 + 3.5 = 7$$
Hash Table

Worst-Case behavior is bad — but what about randomness?

1) **Fix** $h$, our hash, and assume it is good *for all keys*:

   - Simple uniform hash assumption

2) Create a *universal hash function family*:
Simple Uniform Hashing Assumption (SUHA)

Given table of size $m$, a simple uniform hash, $h$, implies

$$\forall k_1, k_2 \in U \text{ where } k_1 \neq k_2, \ Pr(h[k_1] = h[k_2]) = \frac{1}{m}$$

Uniform: Is a flat line $\Rightarrow$ Everything is equally likely to hash to every position

Independent: $\Rightarrow$ Every item hashes independently of every other item
Separate Chaining Under SUHA

Claim: Under SUHA, expected length of chain is $\frac{n}{m}$

$\alpha_j = \text{expected # of items hashing to position } j$

$\alpha_j = \sum_i H_{i,j}$

$H_{i,j} = \begin{cases} 1 & \text{if item } i \text{ hashes to } j \\ 0 & \text{otherwise} \end{cases}$

Table Size: $m$

Num objects: $n$
Separate Chaining Under SUHA

**Claim:** Under SUHA, expected length of chain is \( \frac{n}{m} \)

\( \alpha_j = \text{expected } \# \text{ of items hashing to position } j \)

\[ \alpha_j = \sum_i H_{i,j} \]

\( H_{i,j} = \begin{cases} 
1 & \text{if item } i \text{ hashes to } j \\
0 & \text{otherwise} 
\end{cases} \)

\[ E[\alpha_j] = E\left[ \sum_i H_{i,j} \right] \]
Separate Chaining Under SUHA

**Claim:** Under SUHA, expected length of chain is $\frac{n}{m}$

- $\alpha_j = \text{expected # of items hashing to position } j$

$$\alpha_j = \sum_i H_{i,j}$$

- $H_{i,j} = \begin{cases} 
1 & \text{if item } i \text{ hashes to } j \\
0 & \text{otherwise} \end{cases}$

$$E[\alpha_j] = E\left[\sum_i H_{i,j}\right]$$

$$E[\alpha_j] = \sum_i P(\text{Pr}(H_{i,j} = 1) \times 1 + \text{Pr}(H_{i,j} = 0) \times 0$$
Separate Chaining Under SUHA

**Claim:** Under SUHA, expected length of chain is \( \frac{n}{m} \)

\( \alpha_j \) = expected # of items hashing to position \( j \)

\[
\alpha_j = \sum_i H_{i,j}
\]

\[
H_{i,j} = \begin{cases} 
1 & \text{if item } i \text{ hashes to } j \\
0 & \text{otherwise}
\end{cases}
\]

\[
E[\alpha_j] = \sum_i Pr(H_{i,j} = 1) \cdot 1 + Pr(H_{i,j} = 0) \cdot 0
\]

\[
E[\alpha_j] = n \cdot Pr(H_{i,j} = 1)
\]
Separate Chaining Under SUHA

Claim: Under SUHA, expected length of chain is \( \frac{n}{m} \)

\( \alpha_j \) = expected # of items hashing to position j

\[
\alpha_j = \sum_i H_{i,j}
\]

\[
E[\alpha_j] = E\left[ \sum_i H_{i,j} \right]
\]

\[
E[\alpha_j] = n \cdot Pr(H_{i,j} = 1)
\]

\( H_{i,j} = \begin{cases} 
1 & \text{if item i hashes to j} \\
0 & \text{otherwise} 
\end{cases} \)

\( Pr[H_{i,j} = 1] = \frac{1}{m} \)
Separate Chaining Under SUHA

Claim: Under SUHA, expected length of chain is \( \frac{n}{m} \)

\( \alpha_j = \text{expected # of items hashing to position } j \)

\[ \alpha_j = \sum_i H_{i,j} \]

\[ E[\alpha_j] = E\left[ \sum_i H_{i,j} \right] \]

\[ E[\alpha_j] = n \cdot Pr(H_{i,j} = 1) \]

\[ E[\alpha_j] = \frac{n}{m} \]
Separate Chaining Under SUHA

Under SUHA, a hash table of size $m$ and $n$ elements:

Find runs in: $O(1 + 2)$.

Insert runs in: $O(1)$.

Remove runs in: $O(1 + \alpha)$.

$\alpha = \frac{1}{m}$
Separate Chaining Under SUHA

Pros:

Cons:
Next time: Closed Hashing

**Closed Hashing:** store \( k,v \) pairs in the hash table

\[ S = \{ 1, 8, 15 \} \]

\[ h(k) = k \% 7 \]