Hashing
The 3 main operations on data structures:

SEARCH
INSERT
DELETE

How fast can we do these?
The 3 main operations on data structures:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Complexity</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search</td>
<td>$O(n)$</td>
<td></td>
</tr>
<tr>
<td>Insert</td>
<td>$O(n)$</td>
<td>(if maintaining sorted)</td>
</tr>
<tr>
<td>Delete</td>
<td>$O(n)$</td>
<td>(if we don't want gaps)</td>
</tr>
</tbody>
</table>

How fast can we do these?

- **Search**: $O(n)$, $O(\log n)$ for sorted
- **Insert**: $O(n)$
- **Delete**: $O(n)$, $O(1) + \text{search}$
The 3 main operations on data structures:

- Search
- Insert
- Delete

$O(1)$ expected with assumptions
basic **Hashing**

The 3 main operations on data structures:

- Search
- Insert
- Delete

\[ O(1) \text{ expected with assumptions} \]

- Not "expected worst-case", just "average".
- For some methods, some ops can be \( O(1) \) worst-case.
Hashing

Direct access table: good when keys are distinct & come from a small distribution \( U \).

e.g. \( U = \{0, 1, 2, ..., m-1\} \)
Hashing

Direct access table: good when keys are distinct & come from a small distribution $U$.

e.g. $U = \{0, 1, 2, ..., m-1\}$

Say $m = 74$. Use an array $T$: $012\ldots\phi\phi\ldots\phi\ 73$
Hashing

Direct access table: good when keys are distinct & come from a small distribution $U$.

E.g. $U = \{0, 1, 2, \ldots, m-1\}$

Say $m = 74$. Use an array: $T$

```
0 1 2          \phi \phi         \ldots \phi 73
```

$$\text{search}(T, 2) = 2$$
Direct access table: good when keys are distinct & come from a small distribution $U$.

E.g. $U = \{0, 1, 2, \ldots, m-1\}$

Say $m=74$. Use an array: $T_{0 \ 1 \ 2 \ \cdots \ k \ \phi \ \cdots \ \phi \ 73}$

$\text{search}(T, 2) = 2$  // insert($T, k) \rightarrow T[k] = k$
Direct access table: good when keys are distinct & come from a small distribution $U$.

\[ U = \{0, 1, 2, \ldots, m-1\} \]

Say $m=74$. Use an array: $T = [0, 1, 2, \ldots, \phi, \phi, \ldots, \phi, 73]$.

\[ \text{search}(T, 2) = 2 \quad \text{// insert}(T, k) \rightarrow T[k] = k \quad \text{// delete}(T, k) \rightarrow T[k] = \phi \]

All $\Theta(1)$.
If $U$ is larger than our available storage, $m$
If $U$ is larger than our available storage, $m$ but we are working with a subset $S$ of $U$, where $|S| \leq m$.
If $U$ is larger than our available storage, $m$, but we are working with a subset $S$ of $U$, where $|S| \leq m$.
Example: look up this semester's CS-2413 students using name & academic level (year)

All possible name & year combinations
- Take first letter of name, map to number $L = \{1 \ldots 26\}$
- Map year similarly: sophomore = 2, junior = 3, etc $Y = \{0 \ldots 9\}$

$U$: All possible name & year combinations

$S$: Class list

$T$: Matrix
\[ h \begin{cases} 
& \text{take first letter of name, map to number } \rightarrow L = \{1 \ldots 26\} \\
& \text{map year similarly: sophomore } = 2, \text{ junior } = 3, \text{ etc } \rightarrow Y = \{0 \ldots 9\} \\
& h(\text{student}) = 10 \cdot L + Y \quad \rightarrow \text{unique for any value in } \{L, Y\} 
\end{cases} \]
- Take first letter of name, map to number \( L = \{1 \ldots 26\} \)
- Map year similarly: sophomore = 2, junior = 3, etc \( Y = \{0 \ldots 9\} \)
- \( h(\text{student}) = 10 \cdot L + Y \) → unique for any value in \( \{L, Y\} \)

\( U \)

- CS-2413
- Class list

- Cindy, junior

\( h = 30 + 3 \)

\( T \)

- 0
- 1
- 2
- \( \ldots \)

- Cindy

- 33

- \( m-1 < 270 \)
\[ h \{ \]
\[ \begin{align*}
\cdot & \text{ take first letter of name, map to number } \quad \rightarrow L = \{1 \ldots 26\} \\
\cdot & \text{ map year similarly: sophomore = 2, junior = 3, etc } \quad \rightarrow Y = \{0 \ldots 9\} \\
\cdot & h(\text{student}) = 10L + Y \quad \rightarrow \text{unique for any value in } \{L, Y\}
\end{align*} \]

\[ S : \text{ CS-2413 class list} \]

\[ S : \text{ All possible name \\ & \text{ & year combinations} } \]

\[ h = 30 + 3 \]

\[ \text{PROBLEMS?} \]

\[ 0 \]

\[ 1 \]

\[ 2 \]

\[ 3 \]

\[ \vdots \]

\[ m-1 < 270 \]
\( h \) \{ 
  - take first letter of name, map to number \( L = \{ 1 \ldots 26 \} \)
  - map year similarly: sophomore = 2, junior = 3, etc \( Y = \{ 0 \ldots 9 \} \)
  - \( h(\text{student}) = 10 \cdot L + Y \) → unique for any value in \( \{ L, Y \} \)
\}

\( U \) - class list

CS-2413

\( S \) - class list

Cindy, junior

All possible name & year combinations

\( h = 30 + 3 \)

\( T \)

0

1

2

3

\( m - 1 < 270 \)

PROBLEMS?

(1) Some permanently empty slots
\( h \) {
  \begin{itemize}
    \item take first letter of name, map to number \( L = \{1 \ldots 26\} \)
    \item map year similarly: sophomore = 2, junior = 3, etc \( Y = \{0 \ldots 9\} \)
    \item \( h(\text{student}) = 10 \cdot L + Y \) \( \Rightarrow \) unique for any value in \( \{L, Y\} \)
  \end{itemize}

\( S \):
- CS-2413
- class list

\( U \):
- All possible name & year combinations

\( T \):
- Cindy, junior

\( h = 30 + 3 \)

PROBLEMS?
(1) Some permanently empty slots
(2) Other \( h = 33 \)?
Could use more of the given info to design a more complicated $h()$ and might minimize collisions.
• Could use more of the given info to design a more complicated $h()$

\[ \text{\(\Rightarrow\) might minimize collisions} \]

• But that involves costly processing
• Could use more of the given info to design a more complicated $h()$

  $\Rightarrow$ might minimize collisions

• But that involves costly processing and will need to be repeated if $S$ changes (e.g. next semester)
• Could use more of the given info to design a more complicated \( h() \)
  \[ \mathcal{G} \text{ might minimize collisions} \]

• But that involves costly processing
  and will need to be repeated if \( S \) changes (e.g. next semester)

• We want to keep a simple \( h() \) and deal with collisions
What if many keys map to same slot?

\[ h(65112) = k \]
\[ h(2315) = k \]
\[ h(89) = k \]
What if many keys map to same slot?

**CHAINING**: Make a linked list.

Worst-case time for search, insert, delete?
What if many keys map to same slot?

CHAINING: Make a linked list.

Insert = \( \Theta(1) \)

Search/Delete = \( O(n) \)

\( |S| = n \)
What if many keys map to the same slot?

**CHAINING**: Make a linked list.

- Insert: $\Theta(1)$
- Search/Delete: $O(n)$

Must be consistent for each key.
What if many keys map to same slot?

**CHAINING**: Make a linked list.

- Insert = $\Theta(1)$
- Search/Delete = $O(n)$

Must be consistent for each key

If CHAINING, we don't need $n<m$. 

$|S| = n$
What if many keys map to same slot?

**(CHAINING):** Make a linked list.

- **Insert =** $\Theta(1)$
- **Search/Delete =** $O(n)$

*Must be consistent for each key*

If CHAINING, we don’t need $n < m$.

$n > m$: **COLLISIONS** are **unavoidable**
What if many keys map to same slot?

**CHAINING**: Make a linked list.

- **Insert** = $\Theta(1)$
- **Search/Delete** = $O(n)$

---

If CHAINING, we don't need $n < m$.

$n > m$: **COLLISIONS are unavoidable**

minimize collisions by spreading $S$ into $T$ evenly

→ want random-looking $h()$ yet consistent/deterministic

---

If CHAINING, we don't need $n < m$.
For a random hash function, every slot is equally likely. Probability two given keys collide is \[ \frac{1}{m} \]

Must be consistent for each key. Simple uniform hashing:

**CHAINING**
- Make a linked list.
- Insert = \( \Theta(1) \)
- Search/Delete = \( O(n) \)

What if many keys map to same slot?
What if many keys map to same slot?

**CHAINING:** Make a linked list.

Insert = $\Theta(1)$
Search/Delete = $O(n)$

Must be consistent for each key

For a random $h$, every slot is equally likely : simple uniform hashing

Probability two given keys collide = $\frac{1}{m}$

Average # keys per slot = $\frac{n}{m}$

average size of linked list.
What if many keys map to same slot?

**CHAINING**: Make a linked list.

- **Insert** = $\Theta(1)$
- **Search/Delete** = $O(n)$ (list size)

Must be consistent for each key

For a random $h$, every slot is equally likely: simple uniform hashing

Probability two given keys collide = $\frac{1}{m}$

Average # keys per slot = $\frac{n}{m} = \alpha = \text{"load factor"}$

average size of linked list.
$|S| = n$

$\frac{n}{m} = \alpha = \text{"load factor"}$

average size of linked list.
$|S| = n$

Diagram:

$n/m = \alpha = \text{"load factor"}$

average size of linked list.

Expected time of search (and delete)
$|S| = n$

$\frac{n}{m} = \alpha = \text{"load factor"}$

average size of linked list.

Expected time of search (and delete)

1) $h(\text{key}) \rightarrow \text{slot #} \rightarrow \text{Assume } h() \text{ takes } \Theta(1) \text{ to evaluate}$
\(|S| = n\)

\(\frac{n}{m} = \alpha = \text{"load factor"}\)

average size of linked list.

Expected time of search (and delete)

1) \(h(\text{key}) \rightarrow \text{slot} \#\) \(\rightarrow\) Assume \(h()\) takes \(\Theta(1)\) to evaluate
2) scan list \(\rightarrow\) Expect to scan \(\approx\) half of a list
$|S| = n$

$n/m = \alpha = \text{"load factor"}$

average size of linked list.

Expected time of search (and delete) = $\Theta(1 + \alpha)$

1) $h(\text{key}) \rightarrow \text{slot #} \rightarrow \text{Assume } h() \text{ takes } \Theta(1) \text{ to evaluate}$
2) scan list $\rightarrow \text{Expect to scan } \geq \text{ half of a list}$
$|S| = n$

\[ \frac{n}{m} = \alpha = \text{"load factor"} \]

- average size of linked list.

- great if $\alpha = \Theta(1)$

Expected time of search (and delete) = $\Theta(1+\alpha)$

1) $h(\text{key}) \rightarrow \text{slot #} \rightarrow$ Assume $h(\cdot)$ takes $\Theta(1)$ to evaluate
2) scan list $\rightarrow$ Expect to scan $\geq$ half of a list
$|S| = n$

\[ \frac{n}{m} = \alpha = \text{“load factor”} \]

average size of linked list.

This analysis assumes simple uniform hashing condition

great if $\alpha = O(1)$

**Expected time of search (and delete) = $\Theta(1 + \alpha)$**

1) $h(\text{key}) \rightarrow \text{slot \#}$ → Assume $h()$ takes $\Theta(1)$ to evaluate
2) scan list → Expect to scan $\geq$ half of a list
Choosing Hashing Functions depending on keys and m.
Choosing Hashing Functions depending on keys and m.

Objective: get uniform distribution of keys to slots - always
Choosing Hashing Functions depending on keys and m.

Objective: get uniform distribution of keys to slots - always

Ex: \( h(k) = k \mod m \)

“Division method”

How good is this?
Choosing Hashing Functions depending on keys and m.

Objective: get uniform distribution of keys to slots — always

Ex: \( h(k) = k \mod m \) \begin{align*}
&\text{If } S = \text{integers then it's fine.} \\
&\text{“Division method”} \quad \text{... but if } S = m \cdot i \text{ for } i = 1, 2, 3 \text{ etc. FAIL}
\end{align*}
Choosing Hashing Functions depending on keys and m.

Objective: get uniform distribution of keys to slots - always

Ex: \( h(k) = k \mod m \) \( \iff \) if \( S = \) integers then it's fine.

"Division method" \( \iff \) but if \( S = m \cdot i \) for \( i = 1, 2, 3 \) etc \( \underline{FAIL} \)

We don't want any specific input pattern to affect uniformity.
Choosing Hashing Functions depending on keys and m.

Objective: get uniform distribution of keys to slots - always

Ex: \( h(k) = k \mod m \) If \( S \) = integers then it's fine.

"Division method" ... but if \( S = m \cdot i \) for \( i = 1, 2, 3 \) etc. **FAIL**

We don't want any specific input pattern to affect uniformity.

"Fails" if \( m \) has a small divisor. e.g. for even \( m \), if all keys are even, half of \( T \) empty.
If $m = 2^r$ then $k \mod m = ?$ (does what?)

$r=6 : \quad k = 10110001111011010$
If \( m = 2^r \) then \( k \mod m = k \mod 2^r \) keeps only last \( r \) digits.

\[ r = 6 : \quad k = 1011000111011010 \]
If $m = 2^r$ then $k \mod m = k \mod 2^r$ keeps only last $r$ digits

$r = 6: \quad k = 1011000111011010$

$h$ depends on a small part of the input (key)
If \( m = 2^r \) then \( k \mod m = k \mod 2^r \) keeps only last \( r \) digits

\[ r=6 : \quad k = 1011000111011010 \]

\( h \) depends on a small part of the input (key)

heuristic: choose \( m \) : prime & not close to power of 2
Suppose \( m = 2^r \), and we are using \( w \)-bit words (keys)

\[
h(k) = (A \cdot k) \mod 2^w \text{ right-shifted by } w-r \]

\( \rightarrow \) some odd integer in \([2^{w-r} \ldots 2^{w-1}]\) \(\rightarrow\) \(w\)-bit # with leading 1.

**heuristic**: pick \( A \) not close to any power of 2

\[\text{ex: } m = 2^3 : r = 3 \quad w = 7 \]

\[
A = 1011001 \quad \Rightarrow \quad A \cdot k = 1001010 \quad \boxed{0110011} \quad \text{remains after } \mod 2^7
\]

\[
h(1101011) = 011
\]

If we had \( A = 2^{w-1} \) \( \rightarrow \) \( A \cdot k = 11010110000000 \)

or, \( A = 2^5 \) \( \rightarrow \) \( A \cdot k = 00110101100000 \)

**heuristic provides some "randomness" to the process.**