Hashing
The 3 main operations on data structures:/search/insert/delete

How fast can we do these? (as seen so far in class)
The 3 main operations on data structures: **SEARCH**, **INSERT**, **DELETE**

How fast can we do these? (as seen so far in class)

**SEARCH**
- $O(n)$
- $O(\log n)$ sorted

**INSERT**
- $O(n)$ (if maintaining sorted)

**DELETE**
- $O(n)$ (if we don't want gaps)
- $O(n)$ $O(1) + $ search
The 3 main operations on data structures:

\[ \text{SEARCH} \quad \text{INSERT} \quad \text{DELETE} \quad O(1) \quad \text{expected with assumptions} \]
The 3 main operations on data structures:

SEARCH, INSERT, DELETE

$O(1)$ 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. $U = \{0, 1, 2, \ldots, m-1\}$
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:

\[
\begin{array}{cccccccc}
0 & 1 & 2 & \cdots & \phi & \phi & \cdots & \phi & 73 \\
0 & 1 & 2 & k & m-1
\end{array}
\]
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:

\[
\begin{array}{cccccccc}
0 & 1 & 2 & \cdots & \phi & \phi & \cdots & \phi & 73 \\
0 & 1 & 2 & k & m-1
\end{array}
\]

\[ \text{search}(T, 2) = 2 \]
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: $T_{\{0,1,2,\ldots,73\}}$

Search: $T(2) = 2 \quad \text{// insert}(T, k) \rightarrow T[k] = k$
Hashing

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

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

Say \( m = 74 \). Use an array:

\[
\begin{array}{c}
0 & 1 & 2 & \cdots & \phi & \phi & \cdots & \phi & 73 \\
0 & 1 & 2 & k & m-1
\end{array}
\]

search\((T, 2)\) = 2  
// insert\((T, k)\) \( \rightarrow \) \( T[k] = k \)  
// delete\((T, k)\) \( \rightarrow \) \( T[k] = \emptyset \)

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 $|S| \leq m$. 

$U$: all integers w/ 5 digits

$S$: 65112, 12837, 95617

$T$: 

0
1
2

$\cdots$

$m-1$ (e.g. 73)
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 COMP160 students using name & academic level (year)

\( U \)

\( S \): COMP 160 class list

All possible name & year combinations

\( T \)

0

1

2

\vdots

m-1
- Take first letter of name, map to number \( \rightarrow L = \{1 \ldots 26\} \)
- Map year similarly: sophomore = 2, junior = 3, etc \( \rightarrow Y = \{0 \ldots 9\} \)

\[ U \]

\[ S : \text{COMP 160 class list} \]

All possible name & year combinations

\[ T \]

\[ 0 \]

\[ 1 \]

\[ 2 \]

\[ \vdots \]

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

All possible name & year combinations

\( S \): COMP 160 class list

\( T \)

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

S: COMP 160

U: class list

Chao, junior

All possible name & year combinations

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

\[ h \]

\( S \): COMP 160

\( M \): class list

Chao, junior

\( h = 30 + 3 \)

All possible name & year combinations

PROBLEMS?

\[ 0 \]

\[ 1 \]

\[ 2 \]

\[ \vdots \]

\[ m-1 \]

\[ \leq \]

\[ 270 \]
- 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 \( \{L, Y\} \)

\( U \) all possible name & year combinations

\( S \) COMP 160 class list

Chao, junior

\( h = 30 + 3 \)

PROBLEMS?

1. Some permanently empty slots

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

For example, Chao, junior:
- \( S: \text{COMP 160 class list} \)
- \( h = 30 + 3 \)
- \( T \) has 33 at position 30.

**PROBLEMS?**
1. Some permanently empty slots.
2. Other \( h = 33 \)?
Could use more of the given info to design a more complicated \( h() \)

\( \Rightarrow \) might minimize collisions
• Could use more of the given info to design a more complicated \( h() \) that might minimize collisions.

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

- 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 = $O(n)$

$|S| = n$

$u$
What if many keys map to the same slot? CHAINING: Make a linked list. 

Insert = $\Theta(1)$

Search/Delete = $O(n)$ list size

Must be consistent for each key
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.

If CHAINING, we don't need $n < m$. 
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** = $O(n)$

Must be consistent for each key

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

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

\[
|S| = n
\]

minimize collisions by spreading
- $S$ into $T$ evenly
- want random-looking $h()$
  yet consistent/deterministic
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 = $\frac{1}{m}$
What if many keys map to the same slot?

CHAINING: Make a linked list.

- Insert = $O(1)$
- 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 = $\frac{n}{m}$

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

CHAINING: Make a linked list.

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

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

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

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

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

average size of linked list.

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) \( h(\text{key}) \rightarrow \text{slot\#} \rightarrow \text{Scan list} \)

2) \( \text{Assume } h() \text{ takes } \Theta(1) \text{ to evaluate} \)

Expect to scan \( \approx \) half of a list
Expected time of search (and delete) = $\Theta(1+\alpha)$

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
$|S| = n$

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

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) \( \text{scan list} \rightarrow \text{Expect to scan } \geq \text{half of a list} \)
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: \( h(k) = k \mod m \) \( \{ \) If \( S = \) integers then it's fine.

"Division method" \( \) ...but if \( S = m \cdot i \) for \( i = 1, 2, 3 \) etc **FAIL**
Choosing Hashing Functions depending on keys and \( m \).

Objective: get uniform distribution of keys to slots - always

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

"Division method" \[ \ldots \text{but if } S = \] \( m \cdot i \) \( \text{for } i = 1, 2, 3 \text{ ete} \] \textbf{FAIL}

We don't want any specific input pattern to affect uniformity. (similar to sorting algo not working well on pre-sorted data)
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.

(similar to sorting algo not working well on pre-sorted data)

"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?)

$r = 6 : \quad k = 1011000111011010$
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.

$r=6 : \quad k = \overline{1011000111011010}$

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

heuristic: choose $m$ : prime & not close to power of 2
"Multiplication Method"  

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 \([2^{w-1} \ldots 2^w-1]\) \( \Rightarrow \) \( w \)-bit # with leading 1.

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

ex: \( m = 2^3 ; r = 3 \) \( \quad A = 1011001 \quad \Rightarrow \quad A \cdot k = 1001010 \circled{0110011} \text{ remains after } \mod 2^7 \)

\[
h(1101011) = 011
\]

If we had \( A = 2^{w-1} \) \( \rightarrow \) \( A \cdot k = 110101100000000 \)

or, \( A = 2^5 \) \( \rightarrow \) \( A \cdot k = 001101011000000 \)

\{ heuristic provides some "randomness" to the process. \}