Data Abstraction
Maps (10.1 and 10.2.1)

Janyl Jumadinova

November 18, 2020
A map models a searchable collection of key-value entries.
A map models a searchable collection of key-value entries.

The main operations of a map are for searching, inserting, and deleting items.
Maps

- A **map** models a searchable collection of key-value entries.
- The main operations of a map are for searching, inserting, and deleting items.
- Multiple entries with the same key are not allowed.
Maps

- **A map** models a searchable collection of key-value entries.
- The main operations of a map are for searching, inserting, and deleting items.
- Multiple entries with the same key are not allowed.
- **Applications:**
  - address book
  - student-record database
The Map ADT

- **get**(k): if the map \( M \) has an entry with key \( k \), return its associated value; else, return null.

- **put**(k, v): insert entry \( (k, v) \) into the map \( M \); if key \( k \) is not already in \( M \), then return null; else, return old value associated with \( k \).

- **remove**(k): if the map \( M \) has an entry with key \( k \), remove it from \( M \) and return its associated value; else, return null.

- **size()**, **isEmpty()**, **entrySet()**, **keySet()**, **values()**: return an iterable collection of the entries in \( M \), keys in \( M \), and values in \( M \), respectively.
The Map ADT

- **get(k)**: if the map $M$ has an entry with key $k$, return its associated value; else, return null.

- **put(k, v)**: insert entry $(k, v)$ into the map $M$; if key $k$ is not already in $M$, then return null; else, return old value associated with $k$. 
The **Map ADT**

- **get(k):** if the map \( M \) has an entry with key \( k \), return its associated value; else, return null.
- **put(k, v):** insert entry \((k, v)\) into the map \( M \); if key \( k \) is not already in \( M \), then return null; else, return old value associated with \( k \).
- **remove(k):** if the map \( M \) has an entry with key \( k \), remove it from \( M \) and return its associated value; else, return null.
The Map ADT

- **get(k)**: if the map $M$ has an entry with key $k$, return its associated value; else, return null.

- **put(k, v)**: insert entry $(k, v)$ into the map $M$; if key $k$ is not already in $M$, then return null; else, return old value associated with $k$.

- **remove(k)**: if the map $M$ has an entry with key $k$, remove it from $M$ and return its associated value; else, return null.

- **size()**, **isEmpty()**
The Map ADT

- **get(k)**: if the map $M$ has an entry with key $k$, return its associated value; else, return null.
- **put(k, v)**: insert entry $(k, v)$ into the map $M$; if key $k$ is not already in $M$, then return null; else, return old value associated with $k$.
- **remove(k)**: if the map $M$ has an entry with key $k$, remove it from $M$ and return its associated value; else, return null.
- **size()**, **isEmpty()**
- **entrySet()**: return an iterable collection of the entries in $M$
The Map ADT

- **get(k)**: if the map $M$ has an entry with key $k$, return its associated value; else, return null.

- **put(k, v)**: insert entry $(k, v)$ into the map $M$; if key $k$ is not already in $M$, then return null; else, return old value associated with $k$.

- **remove(k)**: if the map $M$ has an entry with key $k$, remove it from $M$ and return its associated value; else, return null

- **size()**, **isEmpty()**

- **entrySet()**: return an iterable collection of the entries in $M$

- **keySet()**: return an iterable collection of the keys in $M$
The Map ADT

- **get(k)**: if the map \( M \) has an entry with key \( k \), return its associated value; else, return null.
- **put(k, v)**: insert entry \( (k, v) \) into the map \( M \); if key \( k \) is not already in \( M \), then return \( \text{null} \); else, return old value associated with \( k \).
- **remove(k)**: if the map \( M \) has an entry with key \( k \), remove it from \( M \) and return its associated value; else, return \( \text{null} \).
- **size()**, **isEmpty()**
- **entrySet()**: return an iterable collection of the entries in \( M \)
- **keySet()**: return an iterable collection of the keys in \( M \)
- **values()**: return an iterator of the values in \( M \)
## Example

<table>
<thead>
<tr>
<th>Operation</th>
<th>Output</th>
<th>Map</th>
</tr>
</thead>
<tbody>
<tr>
<td>isEmpty()</td>
<td>true</td>
<td>Ø</td>
</tr>
<tr>
<td>put(5,A)</td>
<td>null</td>
<td>(5,A)</td>
</tr>
<tr>
<td>put(7,B)</td>
<td>null</td>
<td>(5,A), (7,B)</td>
</tr>
<tr>
<td>put(2,C)</td>
<td>null</td>
<td>(5,A), (7,B), (2,C)</td>
</tr>
<tr>
<td>put(8,D)</td>
<td>null</td>
<td>(5,A), (7,B), (2,C), (8,D)</td>
</tr>
<tr>
<td>put(2,E)</td>
<td>C</td>
<td>(5,A), (7,B), (2,E), (8,D)</td>
</tr>
<tr>
<td>get(7)</td>
<td>B</td>
<td>(5,A), (7,B), (2,E), (8,D)</td>
</tr>
<tr>
<td>get(4)</td>
<td>null</td>
<td>(5,A), (7,B), (2,E), (8,D)</td>
</tr>
<tr>
<td>get(2)</td>
<td>E</td>
<td>(5,A), (7,B), (2,E), (8,D)</td>
</tr>
<tr>
<td>size()</td>
<td>4</td>
<td>(5,A), (7,B), (2,E), (8,D)</td>
</tr>
<tr>
<td>remove(5)</td>
<td>A</td>
<td>(7,B), (2,E), (8,D)</td>
</tr>
<tr>
<td>remove(2)</td>
<td>E</td>
<td>(7,B), (8,D)</td>
</tr>
<tr>
<td>get(2)</td>
<td>null</td>
<td>(7,B), (8,D)</td>
</tr>
<tr>
<td>isEmpty()</td>
<td>false</td>
<td>(7,B), (8,D)</td>
</tr>
</tbody>
</table>
A Simple List-Based Map

We can implement a map using an unsorted list:
A Simple List-Based Map

We can implement a map using an unsorted list:

- We store the items of the map in a list \( S \) (based on a doublylinked list), in arbitrary order.
The get\((k)\) Algorithm

**Algorithm get\((k)\):**

\[
B = S\text{.positions}() \quad \{ B \text{ is an iterator of the positions in } S \} \\
\textbf{while} \ B\text{.hasNext}() \ \textbf{do} \\
\quad p = B\text{.next}() \quad \{ \text{ the next position in } B \} \\
\quad \textbf{if} \ p\text{.element}()\text{.getKey}() = k \quad \textbf{then} \\
\quad \quad \textbf{return} \ p\text{.element}()\text{.getValue}() \\
\quad \textbf{return null} \quad \{ \text{there is no entry with key equal to } k \}
\]
Algorithm put(k, v):
B = S.positions()
while B.hasNext() do
    p = B.next()
    if p.element().getKey() = k then
        t = p.element().getValue()
        S.set(p, (k, v))
    return t  {return the old value}
S.addLast((k, v))
n = n + 1    {increment variable storing number of entries}
return null  {there was no entry with key equal to k}
Algorithm remove(k):
B = S.positions()
while B.hasNext() do
    p = B.next()
    if p.element().getKey() = k then
        t = p.element().getValue()
        S.remove(p)
        n = n - 1  \{decrement number of entries\}
    return t  \{return the removed value\}
return null  \{there is no entry with key equal to k\}
Performance of a List-Based Map

- **put** takes $O(1)$ time since we can insert the new item at the beginning or at the end of the sequence.
Performance of a List-Based Map

- **put** takes $O(1)$ time since we can insert the new item at the beginning or at the end of the sequence.
- **get** and **remove** take $O(n)$ time since in the worst case (the item is not found) we traverse the entire sequence to look for an item with the given key.
Performance of a List-Based Map

- **put** takes $O(1)$ time since we can insert the new item at the beginning or at the end of the sequence.
- **get** and **remove** take $O(n)$ time since in the worst case (the item is not found) we traverse the entire sequence to look for an item with the given key.
- The unsorted list implementation is effective only for maps of small size or for maps in which puts are the most common operations.
More General Kinds of Keys

What should we do if our keys are not integers in the range from 0 to $N-1$?
More General Kinds of Keys

What should we do if our keys are not integers in the range from 0 to $N-1$?

- Use a **hash function** to map general keys to corresponding indices in a table.
More General Kinds of Keys

What should we do if our keys are not integers in the range from 0 to \( N-1 \)?

- Use a **hash function** to map general keys to corresponding indices in a table.
- For instance, the last four digits of a Social Security number.

![Hash Function Diagram]

Janyl Jumadinova

Data Abstraction  Maps (10.1 and 10.2.1)  November 18, 2020  10 / 11
Hash Functions and Hash Tables

What should we do if our keys are not integers in the range from 0 to $N-1$?

- A **hash function** $h$ maps keys of a given type to integers in a fixed interval $[0, N-1]$. 

**Example:**

$h(x) = x \mod N$ is a hash function for integer keys

The integer $h(x)$ is called the hash value of key $x$.

**Example:**

$h(x) = x \mod N$ is a hash function for integer keys

The integer $h(x)$ is called the hash value of key $x$. 

A hash table for a given key type consists of:

- Hash function $h$
- Array (called table) of size $N$

When implementing a map with a hash table, the goal is to store item $(k, o)$ at index $i = h(k)$. 

Janyl Jumadinova

Data Abstraction Maps (10.1 and 10.2.1)  
November 18, 2020 11 / 11
Hash Functions and Hash Tables

What should we do if our keys are not integers in the range from 0 to \(N-1\)?

- A **hash function** \(h\) maps keys of a given type to integers in a fixed interval \([0, N - 1]\)

- *Example:*
  
  \[ h(x) = x \text{mod} N \]
  
  is a hash function for integer keys.
Hash Functions and Hash Tables

What should we do if our keys are not integers in the range from 0 to $N-1$?

- A **hash function** $h$ maps keys of a given type to integers in a fixed interval $[0, N - 1]$

  *Example:*  
  $h(x) = x \mod N$ is a hash function for integer keys

- The integer $h(x)$ is called the hash value of key $x$
Hash Functions and Hash Tables

What should we do if our keys are not integers in the range from 0 to $N-1$?

- A **hash function** $h$ maps keys of a given type to integers in a fixed interval $[0, N - 1]$.

  *Example:*
  
  $h(x) = x \, \text{mod} \, N$ is a hash function for integer keys.

- The integer $h(x)$ is called the hash value of key $x$.

  *Example:*
  
  $h(x) = x \, \text{mod} \, N$ is a hash function for integer keys.
What should we do if our keys are not integers in the range from 0 to $N-1$?

- **A hash function $h$ maps keys of a given type to integers in a fixed interval $[0, N-1]$**

- **Example:**
  
  $h(x) = x \mod N$ is a hash function for integer keys

- The integer $h(x)$ is called the hash value of key $x$.

- **Example:**
  
  $h(x) = x \mod N$ is a hash function for integer keys

- The integer $h(x)$ is called the hash value of key $x$. 
What should we do if our keys are not integers in the range from 0 to $N-1$?

- A **hash function** $h$ maps keys of a given type to integers in a fixed interval $[0, N-1]$

  *Example:*
  
  $h(x) = x \mod N$ is a hash function for integer keys

- The integer $h(x)$ is called the hash value of key $x$

  *Example:*
  
  $h(x) = x \mod N$ is a hash function for integer keys

- The integer $h(x)$ is called the hash value of key $x$

- A **hash table** for a given key type consists of:
  
  - Hash function $h$
  
  - Array (called table) of size $N$

- When implementing a map with a hash table, the goal is to store item $(k, o)$ at index $i = h(k)$