Ordered Maps & Dictionaries

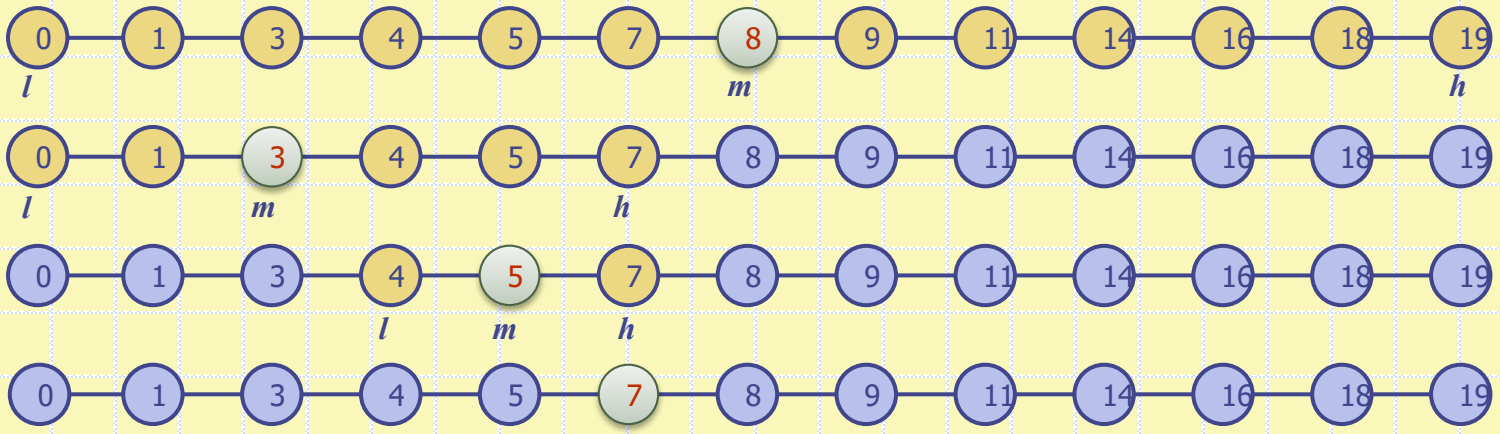
Dr. Aiman Hanna

Department of Computer Science & Software Engineering
Concordia University, Montreal, Canada

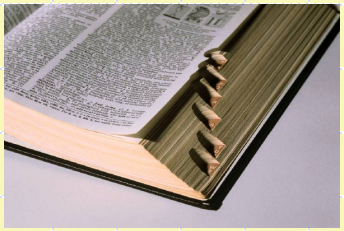
These slides have been extracted, modified and updated from original slides of :
Data Structures and Algorithms in Java, 5th edition. John Wiley& Sons, 2010. ISBN 978-0-470-38326-1.
Data Structures and the Java Collections Framework by William J. Collins, 3rdedition, ISBN 978-0-470-48267-4.
Both books are published by Wiley.

Coverage

- Section 9.3: Ordered Maps & Binary Search
- Section 9.5: Dictionaries
-

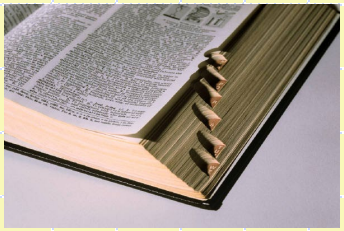


Ordered Maps



- In some applications, simply looking up values based on associated keys is not enough.
- The entries may need to be sorted in the map according in **total order**.
- An ordered map would have the usual operations that a map has, but also maintains an order relation for the keys.
- Keys are assumed to come from a total order; a comparator can be used to provide the needed order between the keys.

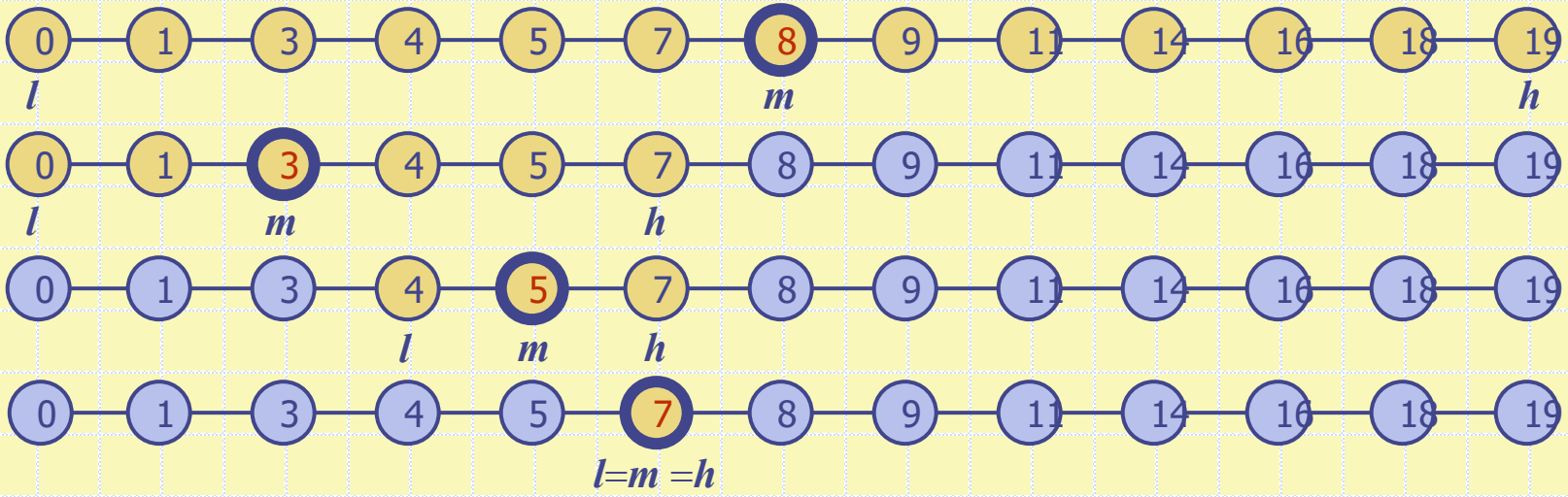
Ordered Maps



- Since the entries are sorted, some additional methods can efficiently be provided:
  - firstEntry(): return the entry with smallest key, or null if the map is empty
  - lastEntry(): return the entry with largest key, or null if the map is empty
  - floorEntry(k): return the entry with the largest key $\leq k$
  - ceilingEntry(k): return the entry with the smallest key $\geq k$
    - These two operations also return null if the map is empty

Binary Search

- Binary search can perform operations get, floorEntry and ceilingEntry on an ordered map implemented by means of an array-based sequence, sorted by key
  - similar to the high-low game
  - at each step, the number of candidate items is halved
  - terminates after O(log n) steps
- Example: find(7)



Performance of Ordered Maps

- We can store the items in an array-based sequence, sorted by key.
- Performance:
  - get, floorEntry and ceilingEntry take $O(\log n)$ time, using binary search
  - put takes $O(n)$ time since in the worst case we have to shift $n/2$ items to make room for the new item
  - remove take $O(n)$ time since in the worst case we have to shift $n/2$ items to compact the items after the removal
- Ordered maps are effective only for small size maps or for maps on which searches are the most common operations, while insertions and removals are rarely performed.

Dictionary ADT

- Like a map, a dictionary stores *key-value* entries.
- Similarly, a dictionary allows the keys and the values to be of any object types.
- However, in contrast to maps, where the keys must be unique, a dictionary allows for multiple entries to have the same keys.
- This is very much like an English dictionary, for instance, which allows for a multiple definitions for the same word.
- The main operations of a dictionary are searching, inserting, and deleting items.

Dictionary ADT

- As an ADT, an (unordered) *dictionary* D supports the following methods:
  - get(k): if the dictionary has an entry with key k, return it, else, return null. If more than one entry exists with key k then *arbitrarily* return one of them.
  - getAll(k): return an iterable collection of all entries with key k.
  - put(k, v): insert an entry with key k and value v and return that entry.
  - remove(e): remove the entry e from the dictionary and return it; error occurs if e is not in the dictionary.
  - entrySet(): return an iterable collection of the entries in the dictionary.
  - size(), isEmpty()

Example

Operation	Output	Dictionary
put(5,A)	(5,A)	(5,A)
put(7,B)	(7,B)	(5,A),(7,B)
put(2,C)	(2,C)	(5,A),(7,B),(2,C)
put(8,D)	(8,D)	(5,A),(7,B),(2,C),(8,D)
put(2,E)	(2,E)	(5,A),(7,B),(2,C),(8,D),(2,E)
get(7)	(7,B)	(5,A),(7,B),(2,C),(8,D),(2,E)
get(4)	**null**	(5,A),(7,B),(2,C),(8,D),(2,E)
get(2)	(2,C)	(5,A),(7,B),(2,C),(8,D),(2,E)
getAll(2)	(2,C),(2,E)	(5,A),(7,B),(2,C),(8,D),(2,E)
size()	5	(5,A),(7,B),(2,C),(8,D),(2,E)
remove(get(5))	(5,A)	(7,B),(2,C),(8,D),(2,E)
get(5)	**null**	(7,B),(2,C),(8,D),(2,E)

A List-Based Dictionary

- A log file or audit trail is a dictionary implemented by means of an unsorted sequence
  - We store the items of the dictionary in a sequence (based on a doubly-linked list or array), in arbitrary order
- Performance:
  - 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 log file is effective only for dictionaries of small size or for dictionaries on which insertions are the most common operations, while searches and removals are rarely performed (e.g., historical record of logins to a workstation)

The getAll and put Algorithms

```
Algorithm getAll(k)
Create an initially-empty list L
for e: D do
    if e.getKey() = k  then
            L.addLast(e)
return L

Algorithm put(k,v)
Create a new entry e = (k,v)
S.addLast(e)   {S is unordered}
return e
```

The remove Algorithm

```
Algorithm remove(e):
{ We don't assume here that e stores its position in S }
B = S.positions()
while B.hasNext() do
    p = B.next()
    if p.element() = e then
        S.remove(p)
        return e
return null      {there is no entry e in D}
```

Hash Table Implementation

- We can also create a hash-table dictionary implementation.
- If we use separate chaining to handle collisions, then each operation can be delegated to a list-based dictionary stored at each hash table cell.
- In that case, the dictionary methods can be achieved in $O(1)$.

Search Table

- A search table is a dictionary implemented by means of a sorted array
  - We store the items of the dictionary in an array-based sequence, sorted by key
  - We use an external comparator for the keys
- Performance:
  - get takes $O(\log n)$ time, using binary search
  - put takes $O(n)$ time since in the worst case we have to shift $n/2$ items to make room for the new item
  - remove takes $O(n)$ time since in the worst case we have to shift $n/2$ items to compact the items after the removal
- A search table is effective only for dictionaries of small size or for dictionaries on which searches are the most common operations, while insertions and removals are rarely performed.