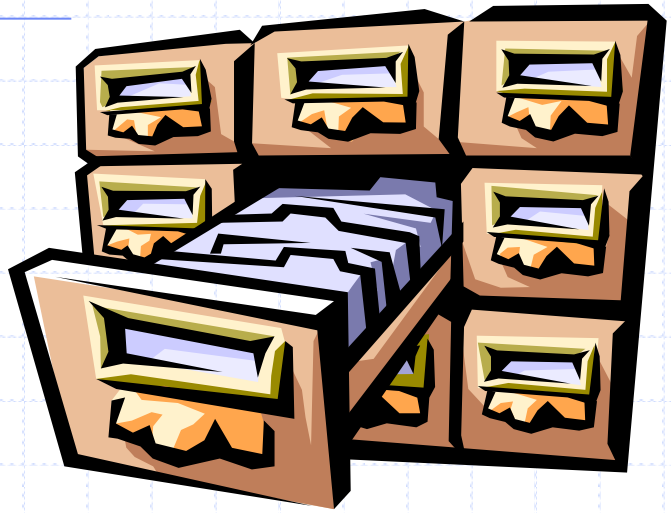
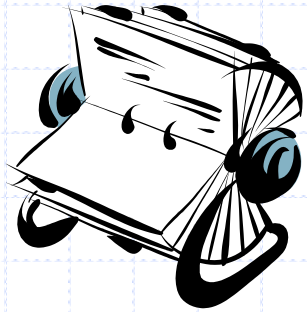
Presentation for use with the textbook Data Structures and Algorithms in Java, 6th edition, by M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, Wiley, 2014

Maps or Dictionaries

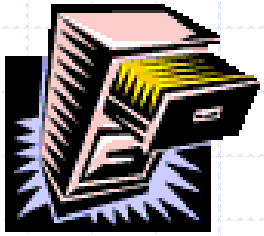


Dictionaries



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

The Dictionary ADT



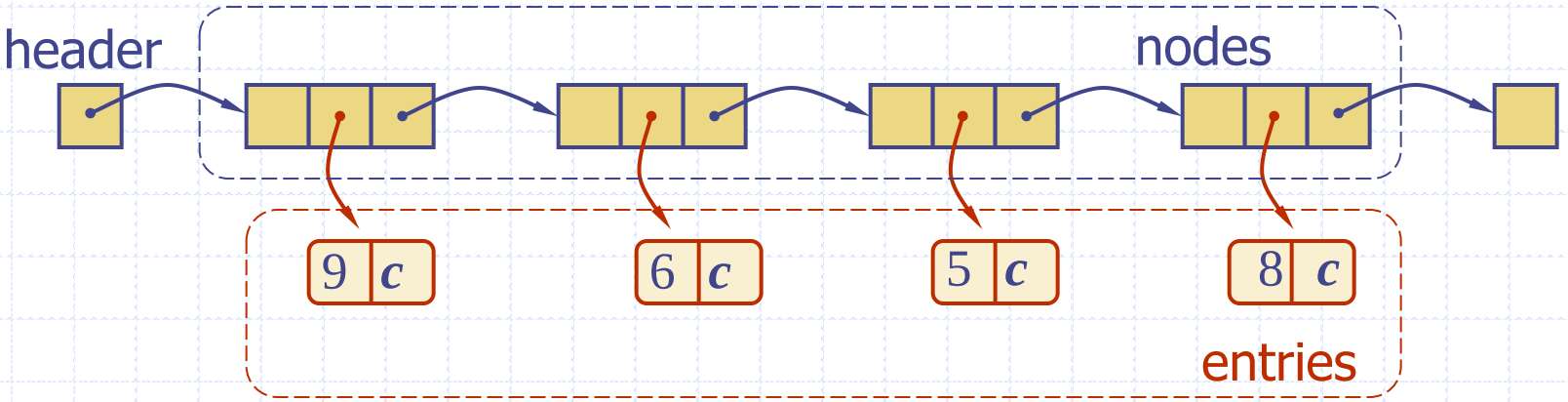
- get(k): if the Dictionary M has an entry with key k, return its associated value; else, return null
- put(k, v): insert entry (k, v) into the Dictionary M; if key k is not already in M, then return null; else, ERROR
- remove(k): if the Dictionary M has an entry with key k, remove it from M and return its associated value; else, ERROR
- 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

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

A Simple List-Based Dictionary

- We can implement a Dictionary using an unsorted list
  - We store the items of the Dictionary in a list S (based on a linked list), in arbitrary order



The get(k) Algorithm

```
Algorithm get(k) {
    p = header
    while p is not null do
        if p.element().getKey() = k then
            return p.element().getValue()
        else p = p.next();
    return null {there is no entry with key equal to k}
}
```

The put(k,v) Algorithm

```
Algorithm put(k,v)
p = header
while p is not null do
    if p.element().getKey() = k  then ERROR
    else p = p.next()
p = new node storing (k,v)
p.setNext (header)
header = p
n = n + 1        {increment variable storing number of entries}
```

The remove(k) Algorithm

```
Algorithm remove(k)
p = header
prev = null
while p is not null do
    if p.element().getKey() = k  then {
        if prev is not null then
            prev.setNext(p.next())
        else header = p.next()
        n = n – 1   {decrement number of entries}
    }
   else {
        prev = p
        p = p.next()
}
```

Performance of a List-Based Dictionary

- Performance:
  - put takes $O(n)$ time since we need to check for duplicated keys
  - 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 Dictionaries of small size.