Dictionaries

Dictionary ADT

- The dictionary ADT models a searchable collection of key-element entries
- The main operations of a dictionary are searching, inserting, and deleting items
- Multiple items with the same key **are** allowed
- Applications:
  - word-definition pairs
  - credit card authorizations
  - DNS mapping of host names (e.g., datastructures.net) to internet IP addresses (e.g., 128.148.34.101)

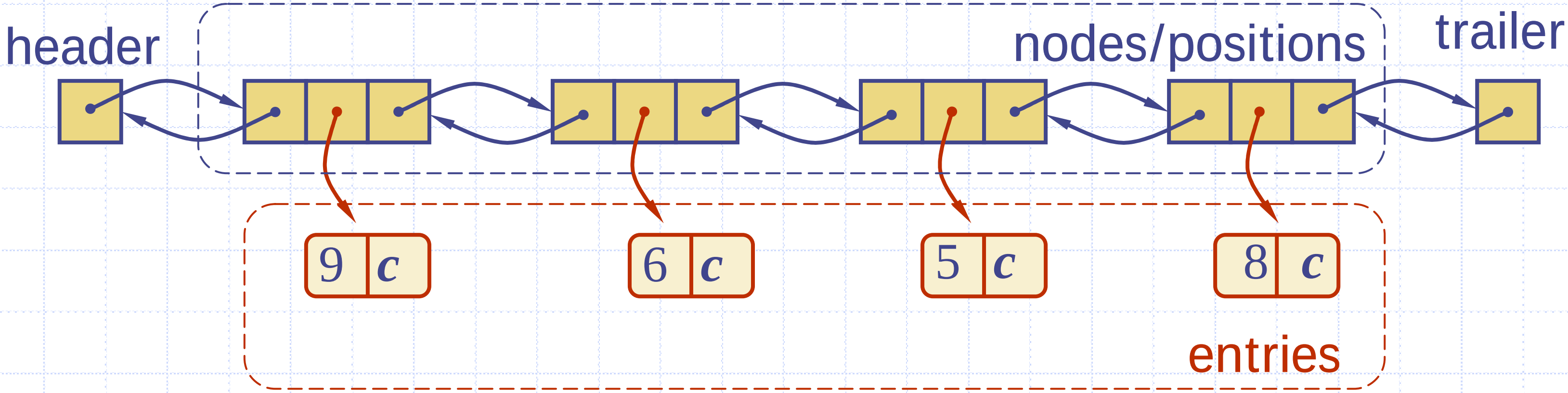
- Dictionary ADT methods:
  - find(k): if the dictionary has an entry with key k, returns it, else, returns null
  - findAll(k): returns an iterator of all entries with key k
  - insert(k, o): inserts and returns the entry (k, o)
  - remove(e): removes the entry e from the dictionary
  - entries(): returns an iterator of the entries in the dictionary
  - size(), isEmpty()

Example

Operation	*Output*	*Dictionary*
insert(5,*A*)	(5,*A*)	(5,*A*)
insert(7,*B*)	(7,*B*)	(5,*A*),(7,*B*)
insert(2,*C*)	(2,*C*)	(5,*A*),(7,*B*),(2,*C*)
insert(8,*D*)	(8,*D*)	(5,*A*),(7,*B*),(2,*C*),(8,*D*)
insert(2,*E*)	(2,*E*)	(5,*A*),(7,*B*),(2,*C*),(8,*D*),(2,*E*)
find(7)	(7,*B*)	(5,*A*),(7,*B*),(2,*C*),(8,*D*),(2,*E*)
find(4)	**null**	(5,*A*),(7,*B*),(2,*C*),(8,*D*),(2,*E*)
find(2)	(2,*C*)	(5,*A*),(7,*B*),(2,*C*),(8,*D*),(2,*E*)
findAll(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(find(5))	(5,*A*)	(7,*B*),(2,*C*),(8,*D*),(2,*E*)
find(5)	**null**	(7,*B*),(2,*C*),(8,*D*),(2,*E*)

A Simple List-Based Dictionary

- We can efficiently implement a dictionary using an unsorted list
  - We store the items of the dictionary in a list L (based on a doubly-linked list), in arbitrary order



The find(k) Algorithm

Algorithm find(*k*.*L*)
In: Key *k* and linked list *L*
Out: node storing *k*, or null if *k* is not in *L*

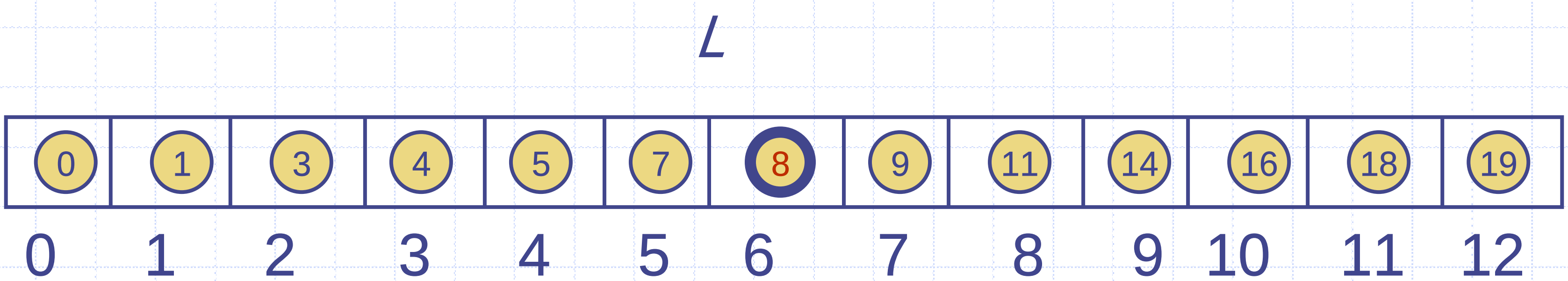
```
p = L.first() {p is the first node in L}
while p != null do
    if p.key() = k then return p
    else p = p.next()
return null {there is no entry with key equal to k}
```

Performance of a List-Based Dictionary

- Performance:
  - insert takes $O(1)$ time since we can insert the new item at the beginning or at the end of the sequence
  - find 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 or for dictionaries in which insertions are the most common operations, while searches and removals are rarely performed (e.g., historical record of logins to a workstation)

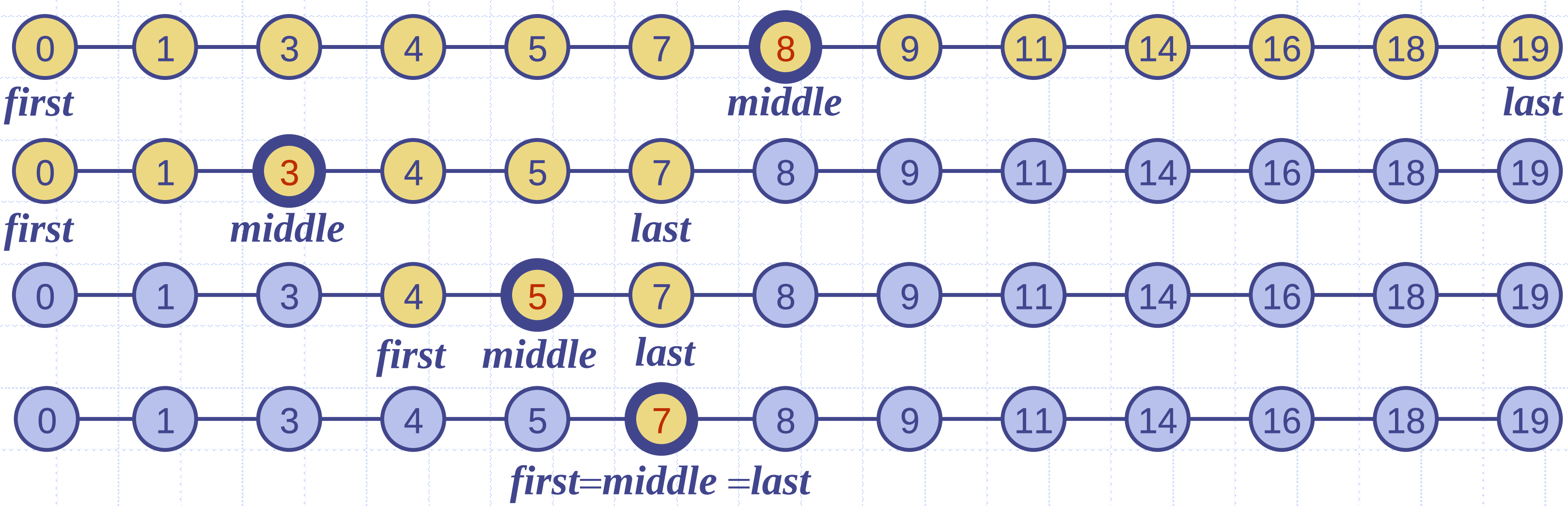
A Sorted List-Based Dictionary

- We can also implement a dictionary using a sorted list
  - We store the items of the dictionary in a list L (based on an array), in non-decreasing order



Binary Search

- Binary search performs operation find(k) on a dictionary implemented by means of an array-based sequence, sorted by key
  - at each step, the number of candidate items is halved
  - terminates after a logarithmic number of steps
- Example: find(7)



The binary search Algorithm

Algorithm find(*k.L,first,last*)

In: Key *k* and array *L[first,last]*

Out: position of *k* in *L*, or -1 if *k* is not in *L*

if *first > last* **then return** -1

else {

middle = *(first+last)*/2

if *(L[middle] = k)* **then return** *middle*

else if *k > L[middle]* **then return** find*(k,L,middle+1,last);*

else return find*(k,L,first,middle-1)*

}

Time complexity

- Let f(n) denote the time complexity of the algorithm when the size of the array is n.
- In the base case the algorithm performs a constant number c’ of operations
- In the recursive case, the algorithm performs a constant number c of operations plus the operations performed in the recursive calls.

  $f(n) = c + f((n-1)/2)$

  $f(0) = c'$

  This system of equations describing the time complexity of the algorithm is called a *recurrence equation*.

Solving the Recurrence Equation

$$f(n) = c + f(\frac{n-1}{2})$$

$$f(n) = c + (c + f(\frac{n-1-2}{2^2}))$$

$$f(n) = c + (c + (c + f(\frac{n-1-2-2^2}{2^3})))$$

$$\vdots$$

$$f(n) = c + (c + \cdots + f(\underbrace{\frac{n-1-2-\cdots-2^i}{2^{i+1}}}_{=0})\cdots)$$

$$f(n) = c + (c + \cdots + (c + f(0))\cdots)$$

$$f(n) = (i+1)c + c', where$$

$$\frac{n-1-2-\cdots-2^i}{2^{i+1}} = 0, so$$

$$n = 1 + 2 + \cdots + 2^i = \sum_{j=0}^{i} 2^j = 2^{i+1} - 1$$

$$Therefore, 2^{i+1} = n + 1 \therefore i = \log(n+1) - 1$$

$$Thus, f(n) = c\log(n+1) + c' = O(\log n)$$

Sorted Array Implementation

- A dictionary can be implemented by means of a sorted array
  - We store the items of the dictionary in an array-based sequence, sorted by key
- Performance:
  - find takes $O(\log n)$ time, using binary search
  - insert takes $O(n)$ time since in the worst case we have to shift n items to make room for the new item
  - remove takes $O(n)$ time since in the worst case we have to shift n items to compact the items after the removal
- This implementation is efficient for dictionaries of small size or for dictionaries on which searches are the most common operations, while insertions and removals are rarely performed (e.g., credit card authorizations)