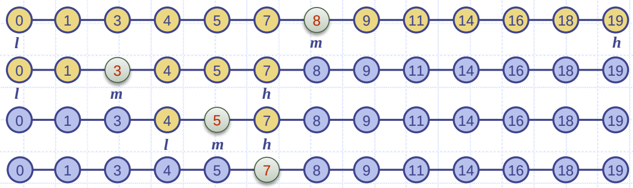
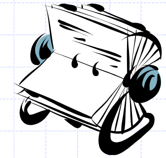
Dictionaries



Dictionary



- 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)

Entry ADT

- An entry stores a key-value pair (k,v)
- Methods:
  - key(): return the associated key
  - value(): return the associated value
  - setKey(k): set the key to k
  - setValue(v): set the value to v

Dictionary ADT

- Dictionary ADT methods:
  - find(k): if there is an entry with key k, returns an iterator to it, else returns the special iterator end
  - findAll(k): returns iterators b and e such that all entries with key k are in the iterator range [b, e) starting at b and ending just prior to e
  - put(k, o): inserts and returns an iterator to it
  - erase(k): remove an entry with key k
  - begin(), end(): return iterators to the beginning and end of the dictionary
  - size(), empty()

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)
find(7)	(7,B)	(5,A),(7,B),(2,C),(8,D),(2,E)
find(4)	**end**	(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)
erase(5)	—	(7,B),(2,C),(8,D),(2,E)
find(5)	**end**	(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
  - find and erase 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 find, put, erase Algorithms

```
Algorithm find(k)
for each p in [S.begin(), S.end()) do
    if p.key() = k  then
        return p

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

Algorithm erase(k):
for each p in [S.begin(), S.end()) do
    if p.key() = k then
        S.erase(p)
```


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.

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:
  - find 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
  - erase 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 (e.g., credit card authorizations)

Binary Search

- Binary search performs operation find(k) on a dictionary 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 a logarithmic number of steps
- Example: find(7)