Symbol tables

Symbol table API

We will have keys like in PQs, but will associate a value with each key

Two operations:

* Insert a value with a key (key-value pair)
* Given a key, search for a corresponding value

This is how a URL (key) connects to an IP address (value)

This is also how you use a word (key) in a dictionary to find a definition (value)

Associative array abstraction

ST() -> Create a table

Void put(Key key, Value val) -> put k/v pair into table; remove if null …. a[key] = val;

Value get(Key key) -> value paired with key (null if k absent)… a[key]

Void delete(Key key) -> remove k/v from table

Boolean contains(Key key) -> is there a value paired with this key

Boolean isEmpty() -> is table empty?

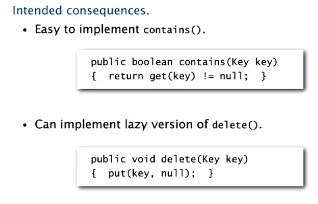
Int size() -> num k/v pairs in table

Iterable<Key> keys() all keys in table

Conventions:

* Values are not null
* Method get() returns null if key not present
* Method put() overwrites old value with new value

Consequences (intended) of conventions:



* Value can be any generic type.
* Keys should be Comparable (compareTo), as they often come from ordered sets (dictionary app, acct nums, etc.)
* If not comparable, can still use equals() to compare
* If no ordering, use hashCode() to add randomness by scrambling key

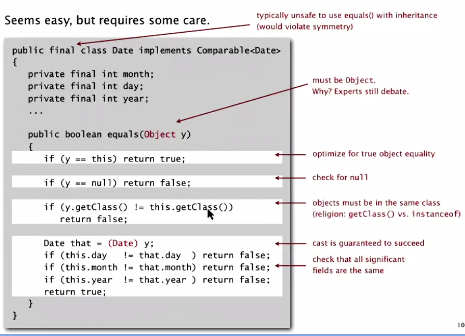
Best practice: immutable types for keys

**Equality test:**

* Reflexive: x.equals(x) is true
* Symmetric: x.equals(y) iff y.equals(x)
* Transitive: if x.equals(y) and y.equals(z), then x.equals(z)
* Non-null: x.equals(null) is false

Default implementation: ( x == y ) ‘do x and y refer to the same object?’

Equals implementation



Recipe:

* Optimize for ref equality
* Check against null
* Check two objects of same type and cast
* Compare each significant field:
  + If field is primitive, use ==
  + If object, use equals() (this is recursive)
  + If array, apply to each entry

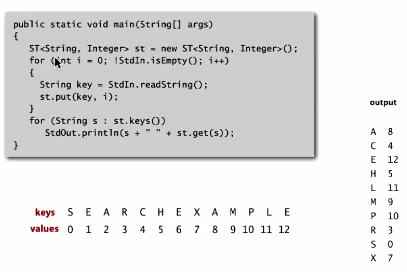
Best practice:

* Don’t compare calculated fields that depend on other fields)
* Compare fields most likely to differ first
* Make compareTo() consistent with equals()

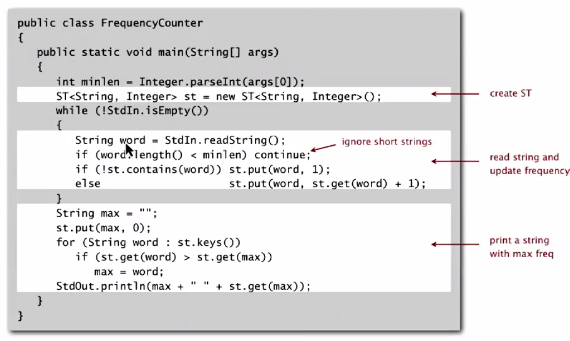
Example ST test client

Associate value *I* with the *ith*string from standard input

In case of multiple, most recent location is the location stored   
(e.g. ‘E’ is 3x, but latest location is 12, so 12 is stored)



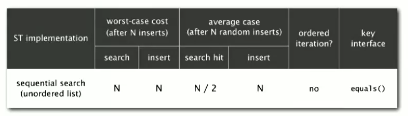
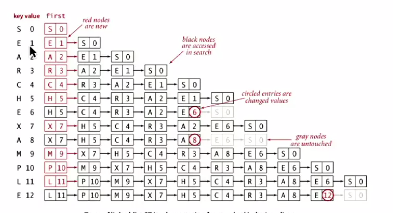
Frequency counter client:



Elementary implementations

Data structure: unordered linked list of k/v pairs

* Search: scan through full list
* Insert: scan through full list

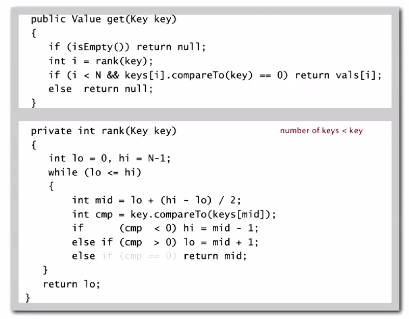


Challenge: more efficient implementations of both search and insert… such as BINARY SORT

Binary search in an ordered array

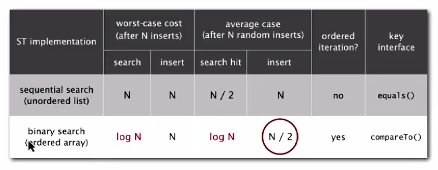
* Parallel arrays (one for k, one for v)
* Array of keys stays sorted, find index associated with the key being searched for

Implementation of binary search with ordered list



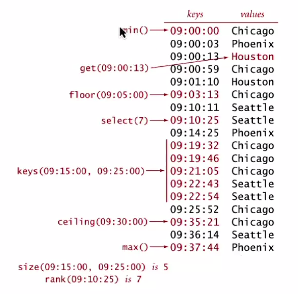
Binary search PROBLEM: each insertion means that all greater keys are shifted over

Overall, this is a much faster solution:

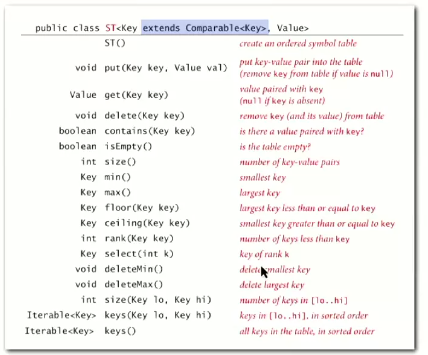


Ordered Operations

Example of ordered symbol table API:



Many methods clients may want to use. API:



Ordered symbol table operations summary

