3.1 Symbol Tables

- API
- elementary implementations
- ordered operations
3.1 SYMBOL TABLES

- API
- elementary implementations
- ordered operations
Why are telephone books (and their cousins) obsolete?

Unsupported phone book operations.

- Add a new name and associated number.
- Remove a given name and associated number.
- Change the number associated with a given name.
Symbol tables

Key–value pair abstraction.
- **Insert** a value with specified key.
- Given a key, **search** for the corresponding value.

**Ex.** DNS lookup.
- Insert domain name with specified IP address.
- Given domain name, find corresponding IP address.

<table>
<thead>
<tr>
<th>domain name</th>
<th>IP address</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.cs.princeton.edu">www.cs.princeton.edu</a></td>
<td>128.112.136.61</td>
</tr>
<tr>
<td>goprincetontigers.com</td>
<td>67.192.28.17</td>
</tr>
<tr>
<td>wikipedia.com</td>
<td>208.80.153.232</td>
</tr>
<tr>
<td>google.com</td>
<td>172.217.11.46</td>
</tr>
</tbody>
</table>
# Symbol table applications

<table>
<thead>
<tr>
<th>application</th>
<th>purpose of search</th>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>dictionary</td>
<td>find definition</td>
<td>word</td>
<td>definition</td>
</tr>
<tr>
<td>book index</td>
<td>find relevant pages</td>
<td>term</td>
<td>list of page numbers</td>
</tr>
<tr>
<td>file share</td>
<td>find song to download</td>
<td>name of song</td>
<td>computer ID</td>
</tr>
<tr>
<td>financial account</td>
<td>process transactions</td>
<td>account number</td>
<td>transaction details</td>
</tr>
<tr>
<td>web search</td>
<td>find relevant web pages</td>
<td>keyword</td>
<td>list of page names</td>
</tr>
<tr>
<td>compiler</td>
<td>find properties of variables</td>
<td>variable name</td>
<td>type and value</td>
</tr>
<tr>
<td>routing table</td>
<td>route Internet packets</td>
<td>destination</td>
<td>best route</td>
</tr>
<tr>
<td>DNS</td>
<td>find IP address</td>
<td>domain name</td>
<td>IP address</td>
</tr>
<tr>
<td>reverse DNS</td>
<td>find domain name</td>
<td>IP address</td>
<td>domain name</td>
</tr>
<tr>
<td>genomics</td>
<td>find markers</td>
<td>DNA string</td>
<td>known positions</td>
</tr>
<tr>
<td>file system</td>
<td>find file on disk</td>
<td>filename</td>
<td>location on disk</td>
</tr>
</tbody>
</table>
Symbol tables: context

Also known as: maps, dictionaries, associative arrays.

Generalizes arrays. Keys need not be integers between 0 and $n - 1$.

Language support.

- External libraries: C, VisualBasic, Standard ML, bash, ...
- Built-in libraries: Java, C#, C++, Scala, ...
- Built-in to language: Awk, Perl, PHP, Tcl, JavaScript, Python, Ruby, Lua.

```python
has_nice_syntax_for_dictionaries['Python'] = True
has_nice_syntax_for_dictionaries['Java'] = False
```

legal Python code
## Basic symbol table API

**Associative array abstraction.** Associate key–value pairs.

```java
class ST<Key extends Comparable<Key>, Value> {
    // create an empty symbol table
    ST();

    // insert key–value pair
    void put(Key key, Value val);

    // value paired with key
    Value get(Key key);

    // all the keys in the symbol table
    Iterable<Key> keys();

    // is there a value paired with key?
    boolean contains(Key key);

    // remove key (and associated value)
    void delete(Key key);

    // is the symbol table empty?
    boolean isEmpty();

    // number of key–value pairs
    int size();
}
```

- Two generic type parameters:
  - `Key` (of type `Comparable<Key>`) for keys
  - `Value` for values

- Methods:
  - `put(key, val)` to insert a key-value pair
  - `get(key)` to retrieve the value paired with a key
  - `keys()` to get all keys
  - `contains(key)` to check if a key exists
  - `delete(key)` to remove a key and its value
  - `isEmpty()` to check if the symbol table is empty
  - `size()` to get the number of key-value pairs
Conventions

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

```
java.util.Map allows null values
```

“Careless use of null can cause a staggering variety of bugs. Studying the Google code base, we found that something like 95% of collections weren’t supposed to have any null values in them, and having those fail fast rather than silently accept null would have been helpful to developers.”

https://code.google.com/p/guava-libraries/wiki/UsingAndAvoidingNullExplained
Key and value types

**Value type.** Any generic type.

**Key type: different assumptions.**
- This lecture: keys are `Comparable`; use `compareTo()`.
- Hashing lecture: keys are any generic type;  
  use `equals()` to test equality and `hashCode()` to scramble key.

**Best practices.** Use immutable types for symbol-table keys.
- Immutable in Java: `String, Integer, Double, Color, ...`
- Mutable in Java: `StringBuilder, Stack, URL, arrays, ...`

“*Classes should be immutable unless there’s a very good reason to make them mutable…. If a class cannot be made immutable, you should still limit its mutability as much as possible.*”

— Joshua Bloch (Java Collections architect)
ST test client for analysis

Frequency counter. Read a sequence of strings from standard input; print one that occurs most often.

```bash
~/Desktop/st> more tinyTale.txt
it was the best of times
it was the worst of times
it was the age of wisdom
it was the age of foolishness
it was the epoch of belief
it was the epoch of incredulity
it was the season of light
it was the season of darkness
it was the spring of hope
it was the winter of despair

~/Desktop/st> java FrequencyCounter 3 < tinyTale.txt
the 10

~/Desktop/st> java FrequencyCounter 8 < tale.txt
business 10

~/Desktop/st> java FrequencyCounter 10 < leipzig1M.txt
government 24763
```

tiny example
(60 words, 20 distinct)

real example
(135,635 words, 10,769 distinct)

real example
(21,191,455 words, 534,580 distinct)
public class FrequencyCounter
{
    public static void main(String[] args)
    {
        int minLength = Integer.parseInt(args[0]);

        ST<String, Integer> st = new ST<>();
        while (!StdIn.isEmpty())
        {
            String word = StdIn.readString();
            if (word.length() < minLength) continue;
            if (!st.containsKey(word)) st.put(word, 1);
            else st.put(word, st.get(word) + 1);
        }

        String champ = "";
        st.put(champ, 0);
        for (String word : st.keys())
        {
            if (st.get(word) > st.get(champ))
                champ = word;
        }
        StdOut.println(champ + " " + st.get(champ));
    }
}
3.1 Symbol Tables

- API
- elementary implementations
- ordered operations
Sequential search in a linked list

Data structure. Maintain an (unordered) linked list of key–value pairs.

Search. Scan through all keys until finding a match.

Insert. Scan through all keys until finding a match; if no match add to front.

get("A")

put("M", 9)

Proposition. In the worst case, search and insert take \(\Theta(n)\) time.
Data structure. Maintain parallel arrays for keys and values, sorted by key.

The tables are:

<table>
<thead>
<tr>
<th>keys[]</th>
<th>vals[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9</td>
<td>8 4 2 5 11 9 10 3 0 7</td>
</tr>
</tbody>
</table>

What are the worst-case running times for **search** and **insert**?

A. $\Theta(\log n)$ and $\Theta(\log n)$

B. $\Theta(n)$ and $\Theta(\log n)$

C. $\Theta(\log n)$ and $\Theta(n)$

D. $\Theta(n)$ and $\Theta(n)$
## Binary search in a sorted array

### Data structure.
Maintain parallel arrays for keys and values, **sorted by key**.

### Search.
Use **binary search** to find key.

### Insert.
Use binary search to find place to insert; shift all larger keys over.

#### get("P")

<table>
<thead>
<tr>
<th>keys[]</th>
<th>vals[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9</td>
<td>0 1 2 3 4 5 6 7 8 9</td>
</tr>
<tr>
<td>A C E H L M P R S Z</td>
<td>8 4 2 5 11 9 10 3 0 7</td>
</tr>
</tbody>
</table>

#### put("P", 10)

<table>
<thead>
<tr>
<th>keys[]</th>
<th>vals[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9</td>
<td>0 1 2 3 4 5 6 7 8 9</td>
</tr>
<tr>
<td>A C E H M R S X - -</td>
<td>8 4 6 5 9 3 0 7 - -</td>
</tr>
</tbody>
</table>
When I first submitted BinarySearchDeluxe.java to TigerFile, the autograder identified a …

A. Correctness bug (false positive or false negative).
B. Performance bug (or infinite loop).
C. Both A and B.
D. Neither A nor B.
**Elementary ST implementations: summary**

<table>
<thead>
<tr>
<th>implementation</th>
<th>guarantee</th>
<th>operations on keys</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>search</td>
<td>insert</td>
</tr>
<tr>
<td>sequential search (unordered list)</td>
<td>$n$</td>
<td>$n$</td>
</tr>
<tr>
<td>binary search (sorted array)</td>
<td>$\log n$</td>
<td>$n$ \dagger</td>
</tr>
</tbody>
</table>

\dagger can do with $\Theta(\log n)$ compares, but still requires $\Theta(n)$ array accesses

**Challenge.** Efficient implementations of both search and insert.
3.1 Symbol Tables

- API
- Elementary implementations
- Ordered operations
Examples of ordered symbol table API

<table>
<thead>
<tr>
<th>keys</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>min()</td>
<td>9:00:00</td>
</tr>
<tr>
<td></td>
<td>9:00:03</td>
</tr>
<tr>
<td></td>
<td>9:00:13</td>
</tr>
<tr>
<td></td>
<td>9:00:59</td>
</tr>
<tr>
<td></td>
<td>9:01:10</td>
</tr>
<tr>
<td></td>
<td>9:03:13</td>
</tr>
<tr>
<td></td>
<td>9:10:11</td>
</tr>
</tbody>
</table>

floor(9:05:00)

<table>
<thead>
<tr>
<th>keys</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>rank(9:10:25) = 7</td>
<td>9:10:25</td>
</tr>
<tr>
<td></td>
<td>9:14:25</td>
</tr>
<tr>
<td></td>
<td>9:19:32</td>
</tr>
<tr>
<td></td>
<td>9:19:46</td>
</tr>
<tr>
<td></td>
<td>9:21:05</td>
</tr>
<tr>
<td></td>
<td>9:22:43</td>
</tr>
<tr>
<td></td>
<td>9:22:54</td>
</tr>
<tr>
<td></td>
<td>9:25:52</td>
</tr>
</tbody>
</table>

select(7)

<table>
<thead>
<tr>
<th>keys</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>ceiling(9:30:00)</td>
<td>9:35:21</td>
</tr>
<tr>
<td></td>
<td>9:36:14</td>
</tr>
<tr>
<td>max()</td>
<td>9:37:44</td>
</tr>
</tbody>
</table>
## Ordered symbol table API

```java
public class ST<Key extends Comparable<Key>, Value>
```

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key min()</td>
<td>smallest key</td>
</tr>
<tr>
<td>Key max()</td>
<td>largest key</td>
</tr>
<tr>
<td>Key floor(Key key)</td>
<td>largest key less than or equal to key</td>
</tr>
<tr>
<td>Key ceiling(Key key)</td>
<td>smallest key greater than or equal to key</td>
</tr>
<tr>
<td>int rank(Key key)</td>
<td>number of keys less than key</td>
</tr>
<tr>
<td>Key select(int k)</td>
<td>key of rank k</td>
</tr>
</tbody>
</table>
```
Rank in a Sorted Array

Problem. Given a sorted array of $n$ distinct keys, find the number of keys strictly less than a given query key.

Q. What if duplicate keys are allowed?
Ordered symbol table operations: summary

<table>
<thead>
<tr>
<th></th>
<th>sequential search</th>
<th>binary search</th>
<th>goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>search</td>
<td>$n$</td>
<td>log $n$</td>
<td>log $n$</td>
</tr>
<tr>
<td>insert</td>
<td>$n$</td>
<td>$n$</td>
<td>log $n$</td>
</tr>
<tr>
<td>min / max</td>
<td>$n$</td>
<td>1</td>
<td>log $n$</td>
</tr>
<tr>
<td>floor / ceiling</td>
<td>$n$</td>
<td>log $n$</td>
<td>log $n$</td>
</tr>
<tr>
<td>rank</td>
<td>$n$</td>
<td>log $n$</td>
<td>log $n$</td>
</tr>
<tr>
<td>select</td>
<td>$n$</td>
<td>1</td>
<td>log $n$</td>
</tr>
</tbody>
</table>

order of growth of worst-case running time for ordered symbol table operations

**Challenge.** Efficient implementations of all operations.