3.1 Symbol Tables

- API
- elementary implementations
- ordered operations
“Smart data structures and dumb code works a lot better than the other way around.” – Eric S. Raymond
3.1 Symbol Tables

- API
  - elementary implementations
  - ordered operations

https://algs4.cs.princeton.edu
Why are telephone books obsolete?

Unsupported operations.

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

key = term
value = article

key = name
value = phone number

key = math function and input
value = function output

key = word
value = definition

key = time and channel
value = TV show
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>128.112.136.11</td>
</tr>
<tr>
<td><a href="http://www.princeton.edu">www.princeton.edu</a></td>
<td>128.112.128.15</td>
</tr>
<tr>
<td><a href="http://www.yale.edu">www.yale.edu</a></td>
<td>130.132.143.21</td>
</tr>
<tr>
<td><a href="http://www.harvard.edu">www.harvard.edu</a></td>
<td>128.103.060.55</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_associative_arrays["Python"] = True
has_nice_syntax_for_associative_arrays["Java"] = False
```
legal Python code
Basic symbol table API

Associative array abstraction. Associate key–value pairs.

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

    ST()  
    void put(Key key, Value val)       insert key–value pair
    Value get(Key key)                 value paired with key
    boolean contains(Key key)          is there a value paired with key?
    Iterable<Key> keys()              all the keys in the symbol table
    void delete(Key key)               remove key (and associated value)
    boolean isEmpty()                 is the symbol table empty?
    int size()                        number of key–value pairs
```

```java
a[key] = val;  
```
Conventions

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

specify Comparable in API
ST test client for analysis

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

```
% 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

% java FrequencyCounter 3 < tinyTale.txt
the 10

% java FrequencyCounter 8 < tale.txt
business 122

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

        String max = "";
        st.put(max, 0);
        for (String word : st.keys())
        {
            if (st.get(word) > st.get(max))
            {
                max = word;
            }
        }
        StdOut.println(max + " " + st.get(max));
    }
}
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 find a match.

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

get("A")

\[
\begin{array}{c}
X & 7 \\
H & 5 \\
C & 4 \\
R & 3 \\
A & 8 \\
E & 6 \\
S & 0 \\
\end{array}
\]

put("M", 9)

\[
\begin{array}{c}
M & 9 \\
X & 7 \\
H & 5 \\
C & 4 \\
R & 3 \\
A & 8 \\
E & 6 \\
S & 0 \\
\end{array}
\]
Elementary ST implementations: summary

<table>
<thead>
<tr>
<th>implementation</th>
<th>guarantee</th>
<th>average case</th>
<th>operations on keys</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>search</td>
<td>insert</td>
<td>search hit</td>
</tr>
<tr>
<td>sequential search</td>
<td>$n$</td>
<td>$n$</td>
<td>$n$</td>
</tr>
<tr>
<td>(unordered list)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Challenge.** Efficient implementations of both search and insert.
Binary search in an ordered array

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

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

**Proposition.** At most $\sim \lg n$ compares to search a sorted array of length $n$.

---

**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 X</td>
<td>8 4 2 5 11 9 10 3 0 7</td>
</tr>
</tbody>
</table>

Loop exits with $lo > hi$: return $7$

Entries in black are $a[lo..hi]$

Entry in red is $a[m]$

$\text{return } vals[6]$
Binary search in an ordered array

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

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

```java
public Value get(Key key) {
    int lo = 0, hi = n-1;
    while (lo <= hi) {
        int mid = lo + (hi - lo) / 2;
        int cmp = key.compareTo(keys[mid]);
        if (cmp < 0) hi = mid - 1;
        else if (cmp > 0) lo = mid + 1;
        else if (cmp == 0) return vals[mid];
    }
    return null; // no matching key
}
```
Binary search: insert

Data structure. Maintain an ordered array of key–value pairs.

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

Proposition. Takes linear time in the worst case.

```
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>
**Elementary ST implementations: summary**

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

$^\dagger$ can do with log $n$ compares, but requires $n$ 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

### min()
09:00:00 - Chicago
09:00:03 - Phoenix
09:00:13 - Houston

### get(09:00:13)
09:00:59 - Chicago
09:01:10 - Houston

### floor(09:05:00)
09:03:13 - Chicago
09:10:11 - Seattle

### select(7)
09:10:25 - Seattle
09:14:25 - Phoenix
09:19:32 - Chicago
09:19:46 - Chicago

### keys(09:15:00, 09:25:00)
09:21:05 - Chicago
09:22:43 - Seattle
09:22:54 - Seattle
09:25:52 - Chicago

### ceiling(09:30:00)
09:35:21 - Chicago
09:36:14 - Seattle

### max()
09:37:44 - Phoenix

size(09:15:00, 09:25:00) is 5
rank(09:10:25) is 7
## 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><code>Key min()</code></td>
<td>smallest key</td>
</tr>
<tr>
<td><code>Key max()</code></td>
<td>largest key</td>
</tr>
<tr>
<td><code>Key floor(Key key)</code></td>
<td>largest key less than or equal to key</td>
</tr>
<tr>
<td><code>Key ceiling(Key key)</code></td>
<td>smallest key greater than or equal to key</td>
</tr>
<tr>
<td><code>int rank(Key key)</code></td>
<td>number of keys less than key</td>
</tr>
<tr>
<td><code>Key select(int k)</code></td>
<td>key of rank k</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Problem.** Given a sorted array of $n$ distinct keys, find the number of keys strictly less than a given query key.
# Binary search: ordered symbol table operations summary

<table>
<thead>
<tr>
<th></th>
<th>Sequential search</th>
<th>Binary search</th>
</tr>
</thead>
<tbody>
<tr>
<td>search</td>
<td>$n$</td>
<td>$\log n$</td>
</tr>
<tr>
<td>insert</td>
<td>$n$</td>
<td>$n$</td>
</tr>
<tr>
<td>min / max</td>
<td>$n$</td>
<td>1</td>
</tr>
<tr>
<td>floor / ceiling</td>
<td>$n$</td>
<td>$\log n$</td>
</tr>
<tr>
<td>rank</td>
<td>$n$</td>
<td>$\log n$</td>
</tr>
<tr>
<td>select</td>
<td>$n$</td>
<td>1</td>
</tr>
</tbody>
</table>

Order of growth of the running time for ordered symbol table operations