Symbol tables

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

Ex. DNS lookup.
- Insert URL with specified IP address.
- Given URL, find corresponding IP address.

<table>
<thead>
<tr>
<th>URL</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>
<tr>
<td><a href="http://www.simpsons.com">www.simpsons.com</a></td>
<td>209.052.165.60</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 given URL</td>
<td>URL</td>
<td>IP address</td>
</tr>
<tr>
<td>reverse DNS</td>
<td>find URL given IP address</td>
<td>IP address</td>
<td>URL</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>

Conventions

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

Intended consequences.

- Easy to implement `contains()`.
  ```java
  public boolean contains(Key key)  
  {  return get(key) != null;  }
  ```

- Can implement lazy version of `delete()`.
  ```java
  public void delete(Key key)  
  {  put(key, null);  }
  ```

Basic symbol table API

**Associative array abstraction.** Associate one value with each key.

```java
public class ST<Key, Value> {
  ST()  
  create a symbol table
  void put(Key key, Value val) 
  put key-value pair into the table 
  (remove key from table if value is null)
  Value get(Key key) 
  value paired with key  
  (null if key is absent)
  void delete(Key key)  
  remove key (and its value) from table
  boolean contains(Key key)  
  is there a value paired with key?  
  boolean isEmpty()  
  is the table empty?  
  int size()  
  number of key-value pairs in the table
  Iterable<Key> keys()  
  all the keys in the table
```

Keys and values

**Value type.** Any generic type.

**Key type: several natural assumptions.**

- Assume keys are `Comparable`, use `compareTo()`.
- Assume keys are any generic type, use `equals()` to test equality.
- Assume keys are any generic type, use `equals()` to test equality; use `hashCode()` to scramble key.

**Best practices.** Use immutable types for symbol table keys.

- Immutable in Java: `String`, `Integer`, `Double`, `java.io.File`, ...
- Immutable in Java: `String`, `Integer`, `Double`, `java.io.File`, ...
- Mutable in Java: `StringBuilder`, `java.net.URL`, arrays, ...

Inbuilt to Java:

- `Comparable` built-in to Java (`stay tuned`)
Equality test

All Java classes inherit a method `equals()`.

Java requirements. For any references `x`, `y` and `z`:
- 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)`

Customized implementations. Integer, Double, String, File, URL, ...

User-defined implementations. Some care needed.

Implementing equals for user-defined types

Seems easy.

```java
public class Date implements Comparable<Date> {
    private final int month;
    private final int day;
    private final int year;
    ...
    public boolean equals(Date that) {
        if (this.day != that.day) return false;
        if (this.month != that.month) return false;
        if (this.year != that.year) return false;
        return true;
    }
}
```

Implementing equals for user-defined types

Seems easy, but requires some care.

```java
public final class Date implements Comparable<Date> {
    private final int month;
    private final int day;
    private final int year;
    ...
    public boolean equals(Object y) {
        if (y == this) return true;
        if (y == null) return false;
        if (y.getClass() != this.getClass())
            return false;
        Date that = (Date) y;
        if (this.month != that.month) return false;
        if (this.year != that.year) return false;
        return true;
    }
}
```

Equals design

"Standard" recipe for user-defined types.
- Optimization for reference equality.
- Check against null.
- Check that two objects are of the same type and cast.
- Compare each significant field:
  - if field is a primitive type, use `==`
  - if field is an object, use `equals()`
  - if field is an array, apply to each entry
- Apply rule recursively
- Alternatively, use `Arrays.equals(a, b)` or `Arrays.deepEquals(a, b)`, but not `a.equals(b)`

Best practices.
- No need to use calculated fields that depend on other fields.
- Compare fields mostly likely to differ first.
- Only use necessary fields, e.g. a webpage is best defined by URL, not number of views.
- Make `compareTo()` consistent with `equals()`.
ST test client for traces

Build ST by associating value $i$ with $i$th string from standard input.

```java
public static void main(String[] args) {
    ST<String, Integer> st = new ST<String, Integer>();
    for (int i = 0; !StdIn.isEmpty(); i++)
    {
        String key = StdIn.readString();
        st.put(key, i);
    }
    for (String s : st.keys())
        StdOut.println(s + " " + st.get(s));
}
```

The order of output depends on the underlying data structure!

<table>
<thead>
<tr>
<th>keys</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>S E A R C H E X A M P L E</td>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12</td>
</tr>
</tbody>
</table>

ST test client for analysis

Frequency counter. Read a sequence of strings from standard input and print out one that occurs with highest frequency.

```java
public class FrequencyCounter {
    public static void main(String[] args) {
        int minlen = Integer.parseInt(args[0]);
        ST<String, Integer> st = new ST<String, Integer>();
        while (!StdIn.isEmpty())
        {
            String word = StdIn.readString();
            if (word.length() < minlen) 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));
    }
}
```

Frequency counter implementation

Create ST
Ignore short strings
Read string and update frequency
Print a string with max freq

| tiny example (60 words, 20 distinct) |
| real example (135,635 words, 10,769 distinct) |
| real example (21,191,455 words, 534,580 distinct) |
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.

Elementary ST implementations: summary

<table>
<thead>
<tr>
<th>ST implementation</th>
<th>worst case cost (after N inserts)</th>
<th>average case (after N random inserts)</th>
<th>ordered iteration?</th>
<th>key interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>sequential search (unordered list)</td>
<td>N</td>
<td>N/2</td>
<td>N</td>
<td>equals()</td>
</tr>
</tbody>
</table>

Challenge. Efficient implementations of both search and insert.

Elementary ST implementations: summary

<table>
<thead>
<tr>
<th>ST implementation</th>
<th>worst case search</th>
<th>average case search hit</th>
<th>ordered iteration?</th>
<th>operations on keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>sequential search (unordered list)</td>
<td>N</td>
<td>N/2</td>
<td>N</td>
<td>equals()</td>
</tr>
</tbody>
</table>

Grey data points are observed costs for i-th operation, reds are their averages

Challenge. Efficient implementations of both search and insert.

Binary search

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

Rank helper function. How many keys < k?

successful search for P

0 9 4 A C E H L M P R S X
5 9 7 A C E H L M P R S X
5 6 5 A C E H L M P R S X
6 6 6 A C E H L M P R S X

entry in black are \( a[i...hi] \)

entries in red are \( a[\ast] \)

unsuccessful search for Q

0 9 4 A C E H L M P R S X
5 9 7 A C E H L M P R S X
5 6 5 A C E H L M P R S X
7 6 6 A C E H L M P R S X

loop exits with \( i > hi \); return 7

Trace of binary search for rank in an ordered array
Binary search: Java implementation

```java
public Value get(Key key)
{
    if (isEmpty()) return null;
    int i = rank(key);
    if (i < N && keys[i].compareTo(key) == 0) return vals[i];
    else return null;
}

private int rank(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 mid;
    }
    return lo;
}
```

Binary search: mathematical analysis

**Proposition.** Binary search uses \( \sim \lg N \) compares to search any array of size \( N \).

**Pf.**
\[
T(N) = \text{number of compares to binary search in a sorted array of size } N.
\]

Recall lecture 2.

Elementary ST implementations: frequency counter

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>1</td>
</tr>
<tr>
<td>S</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>2</td>
</tr>
<tr>
<td>A</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
</tr>
<tr>
<td>H</td>
<td>5</td>
</tr>
<tr>
<td>E</td>
<td>6</td>
</tr>
<tr>
<td>X</td>
<td>7</td>
</tr>
<tr>
<td>A</td>
<td>8</td>
</tr>
<tr>
<td>M</td>
<td>9</td>
</tr>
<tr>
<td>P</td>
<td>10</td>
</tr>
<tr>
<td>L</td>
<td>11</td>
</tr>
<tr>
<td>E</td>
<td>12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>1</td>
</tr>
<tr>
<td>S</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>2</td>
</tr>
<tr>
<td>A</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
</tr>
<tr>
<td>H</td>
<td>5</td>
</tr>
<tr>
<td>E</td>
<td>6</td>
</tr>
<tr>
<td>X</td>
<td>7</td>
</tr>
<tr>
<td>A</td>
<td>8</td>
</tr>
<tr>
<td>M</td>
<td>9</td>
</tr>
<tr>
<td>P</td>
<td>10</td>
</tr>
<tr>
<td>L</td>
<td>11</td>
</tr>
<tr>
<td>E</td>
<td>12</td>
</tr>
<tr>
<td>A</td>
<td>13</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>key value</th>
<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>
<td>0 1 2 3 4 5 6 7 8 9</td>
</tr>
<tr>
<td>S E A C H R S X P L E</td>
<td>1 2 3 4 5 6 7 8 9</td>
<td>1 2 3 4 5 6 7 8 9</td>
</tr>
</tbody>
</table>

Elementary ST implementations: frequency counter

- Costs for java FrequencyCounter 8 < tale.txt using SequentialSearchST
- Costs for java FrequencyCounter 8 < tale.txt using BinarySearchST
Elementary ST implementations: summary

<table>
<thead>
<tr>
<th>ST implementation</th>
<th>worst-case cost (after N inserts)</th>
<th>average case (after N random inserts)</th>
<th>ordered insertion?</th>
<th>key interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>search</td>
<td>N</td>
<td>N</td>
<td>no</td>
<td>equals()</td>
</tr>
<tr>
<td>insert</td>
<td>N</td>
<td>N / 2</td>
<td>yes</td>
<td>compareTo()</td>
</tr>
</tbody>
</table>

### Challenge
Efficient implementations of both search and insert.

API
- Elementary implementations
- Ordered operations

Ordered symbol table API (Example Operations)

<table>
<thead>
<tr>
<th>key</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>min()</td>
<td>09:00:00</td>
</tr>
<tr>
<td></td>
<td>Chicago</td>
</tr>
<tr>
<td>09:00:03</td>
<td>Phoenix</td>
</tr>
<tr>
<td>09:00:11</td>
<td>Houston</td>
</tr>
<tr>
<td>get(09:00:13)</td>
<td>Chicago</td>
</tr>
<tr>
<td>09:00:59</td>
<td>Chicago</td>
</tr>
<tr>
<td>09:01:10</td>
<td>Houston</td>
</tr>
<tr>
<td>floor(09:05:00)</td>
<td>Chicago</td>
</tr>
<tr>
<td>09:03:13</td>
<td>Chicago</td>
</tr>
<tr>
<td>09:10:11</td>
<td>Seattle</td>
</tr>
<tr>
<td>select(7)</td>
<td>Seattle</td>
</tr>
<tr>
<td>09:10:23</td>
<td>Seattle</td>
</tr>
<tr>
<td>09:14:25</td>
<td>Phoenix</td>
</tr>
<tr>
<td>09:19:32</td>
<td>Chicago</td>
</tr>
<tr>
<td>09:19:46</td>
<td>Chicago</td>
</tr>
<tr>
<td>keys(09:15:00, 09:25:00)</td>
<td></td>
</tr>
<tr>
<td>09:21:05</td>
<td>Chicago</td>
</tr>
<tr>
<td>09:22:43</td>
<td>Seattle</td>
</tr>
<tr>
<td>09:22:54</td>
<td>Seattle</td>
</tr>
<tr>
<td>09:25:52</td>
<td>Chicago</td>
</tr>
<tr>
<td>ceiling(09:30:00)</td>
<td>Chicago</td>
</tr>
<tr>
<td>09:35:23</td>
<td>Chicago</td>
</tr>
<tr>
<td>09:36:14</td>
<td>Seattle</td>
</tr>
<tr>
<td>max()</td>
<td>09:37:44</td>
</tr>
<tr>
<td>Phoenix</td>
<td></td>
</tr>
</tbody>
</table>

Examples of ordered symbol-table operations

Ordered symbol table API

```java
public class ST<Key extends Comparable<Key>, Value>
{
    // create an ordered symbol table
    ST()
    
    // put key-value pair into the table (remove key from table if value is null)
    void put(Key key, Value val)
    
    // value paired with key
    Value get(Key key)
    
    // remove key (and its value) from table
    void delete(Key key)
    
    // is there a value paired with key?
    boolean contains(Key key)
    
    // is the table empty?
    boolean isEmpty()
    
    // number of key-value pairs
    int size()
    
    // smallest key
    Key min()
    
    // largest key
    Key max()
    
    // largest key less than or equal to key
    Key floor(Key key)
    
    // smallest key greater than or equal to key
    Key ceiling(Key key)
    
    // number of keys less than key
    int rank(Key key)
    
    // key of rank k
    Key select(int k)
    
    // delete smallest key
    void deleteMin()
    
    // delete largest key
    void deleteMax()
    
    // smallest key in [lo, hi]
    int size(Key lo, Key hi)
    
    // keys in [lo, hi], in sorted order
    Iterable<Key> keys(Key lo, Key hi)
    
    // all keys in the table, in sorted order
    Iterable<Key> keys()
}
```

Examples of ordered symbol-table operations
## Binary search: ordered symbol table operations summary

<table>
<thead>
<tr>
<th>Operation</th>
<th>Sequential Search</th>
<th>Binary Search</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search</td>
<td>$N$</td>
<td>$\lg N$</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>$\lg N$</td>
</tr>
<tr>
<td>Rank</td>
<td>$N$</td>
<td>$\lg N$</td>
</tr>
<tr>
<td>Select</td>
<td>$N$</td>
<td>$1$</td>
</tr>
<tr>
<td>Ordered Iteration</td>
<td>$N \log N$</td>
<td>$N$</td>
</tr>
</tbody>
</table>

*order of growth of the running time for ordered symbol table operations*

The Problem: Insert Operation