Symbol Tables

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.

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>look up word</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>value and type</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>

Symbol table API

Associative array abstraction. Associate one value with each key.

```
public class ST<Key, Value>
{
    ST() // create a symbol table
    void put(Key key, Value val) // put key-value pair into the table
    Value get(Key key) // return value paired with key
    boolean contains(Key key) // is there a value paired with key?
    void remove(Key key) // remove key-value pair from table
    Iterator<Key> iterator() // iterator through keys in 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 `remove()`.

```java
public boolean remove(Key key)
{  put(key, null);          }
```

**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 and `hashCode()` to scramble key.

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

- Immutable in Java: `String`, `Integer`, `BigInteger`, ...
- Mutable in Java: `Date`, `GregorianCalendar`, `StringBuilder`, ...

**ST test client**

Build ST by associating value `i` with ith command-line argument.

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

**Elementary ST implementations**

- Sequential search.
- Binary search.
- Array vs. linked list.

**Why study elementary implementations?**

- Performance benchmarks.
- API details need to be worked out.
- Basis for advanced implementations.
- Method of choice can be one of these in many situations.

**Remark.** Always good practice to study elementary implementations.
Java conventions for $equals()$

All Java objects implement a method $equals()$.

Default implementation: $(x == y)$

Equivalence relation. 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.

Customized implementations. String, URL, Integer, ...

User-defined implementations. Some care needed.

Implementing $equals()$

Seems easy.

```java
public class PhoneNumber
{
    private final int area, exch, ext;
    ...
    public boolean equals(PhoneNumber y)
    {
        PhoneNumber that = (PhoneNumber) y;
        return (this.area == that.area) &&
                (this.exch == that.exch) &&
                (this.ext == that.ext);
    }
}
```

Seems easy, but requires some care.

```java
public final class PhoneNumber
{
    private final int area, exch, ext;
    ...
    public boolean equals(Object y)
    {
        if (y == this) return true;
        if (y == null) return false;
        if (y.getClass() != this.getClass())
            return false;
        PhoneNumber that = (PhoneNumber) y;
        return (this.area == that.area) &&
                (this.exch == that.exch) &&
                (this.ext == that.ext);
    }
}
```

If I'm executing this code, I'm not optimize for true object equality

no safe way to use $equals()$ with inheritance

must be Object. Why? Experts still debate.

objects must be in the same class

If I'm executing this code, I'm not null

optimize for true object equality

objects must be in the same class
Maintain a linked list with keys and values.

Inner node class.
• Instance variable key holds the key.
• Instance variable val holds the value.

Instance variable(s):
• Node first refers to the first node in the list.

```java
class Node {
    private Key key;
    private Value val;
    private Node next;
    public Node(Key key, Value val, Node next) {
        this.key = key;
        this.val = val;
        this.next = next;
    }
}
```

```java
public class UnorderedLinkedST<Key, Value> {
    private Node first;
    public void put(Key key, Value val) {
        for (Node x = first; x != null; x = x.next)
            if (key.equals(x.key)) {  x.val = val; return; }
        first = new Node(key, val, first);
    }
    public Val get(Key key) {
        for (Node x = first; x != null; x = x.next)
            if (key.equals(x.key))
                return x.val;
        return null;
    }
}
```
Unordered linked-list ST: trace

Trace of linked-list ST implementation for standard indexing client

Maintain two parallel arrays with keys and values.

Instance variables.
- keys[i] holds the ith smallest key.
- vals[i] holds the value associated with the ith smallest key.
- \( N \) holds the number of entries.

import java.util.Iterator;
public class UnorderedLinkedST<Key, Value> implements Iterable<Key> {
    ...
    public Iterator<Key> iterator() { return new ListIterator(); }
    private class ListIterator implements Iterator<Key> {
        private Node current = first;
        public boolean hasNext() { return current != null; }
        public void remove() { }
        public Key next() {
            Key key = current.key;
            current = current.next;
            return key;
        }
    }
}

ST implementations: summary

<table>
<thead>
<tr>
<th>ST implementation</th>
<th>worst case</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>unordered array</td>
<td>N</td>
<td>N</td>
<td>N/2</td>
</tr>
<tr>
<td>unordered list</td>
<td>N</td>
<td>N</td>
<td>N/2</td>
</tr>
</tbody>
</table>

Challenge. Efficient implementations of search and insert.

Iterators

Goal. Allow client to iterate over the symbol table keys.
Iterable ST client: frequency counter

Goal. Read a sequence of strings from standard input and print out the number of times each string appears.

```
% java FrequencyCount < tiny.txt
2 age
1 best
1 foolishness
4 it
4 of
2 times
4 was
1 wisdom
1 worst

% java FrequencyCount < tiny.txt
1 foolishness
1 wisdom
2 age
1 worst
2 times
4 of
4 was
4 it
```

Remark. No requirement that keys are iterated in natural order.

- Not in basic API.
- Not a requirement for some clients.
- Not a problem if postprocessing, e.g. with sort or grep.

ST implementations: summary

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

Challenge. Efficient implementations of search, insert, and ordered iteration.

```
public class FrequencyCount
{
   public static void main(String[] args)
   {
      ST<String, Integer> st = new ST<String, Integer>();
      while (!StdIn.isEmpty())
      {
         String key = StdIn.readString();
         if (!st.contains(key)) st.put(key, 1);
         else                   st.put(key, st.get(key) + 1);
      }
      for (String s: st)
         StdOut.println(st.get(s) + " " + s);
   }
}
```

Iterable ST client: A problem?

```
% more tiny.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
% java FrequencyCount < tiny.txt
2 age
1 best
1 foolishness
4 it
4 of
2 times
4 was
1 wisdom
1 worst
```

With one ST implementation

```
% java FrequencyCount < tiny.txt
1 foolishness
1 wisdom
2 age
1 worst
2 times
4 of
4 was
4 it
```

With UnorderedLinkedListST

```
% java FrequencyCount < tiny.txt
1 foolishness
1 wisdom
2 age
1 worst
2 times
4 of
4 was
4 it
```
Ordered array ST implementation

Assumption. Keys are Comparable.

Instance variables.
- keys[i] holds the ith smallest key.
- vals[i] holds the value associated with the ith smallest key.
- N holds the number of entries.

Main reasons to consider using ordered arrays.
- Provides ordered iteration (for free).
- Can use binary search to significantly speed up search.

Unordered array ST implementation (skeleton)

```java
public class OrderedArrayST<Key extends Comparable<Key>, Value> {
    private Value[] vals; // parallel arrays lead to cleaner code
    private Key[] keys; // than defining a type for entries
    private int N;

    public OrderedArrayST(int capacity) {
        keys = (Key[]) new Comparable[capacity];
        vals = (Value[]) new Object[capacity];
    }

    public boolean isEmpty() {
        return N == 0;
    }

    public void put(Key key, Value val) {
        /* see next slides */
    }

    public Value get(Key key) {
        /* see next slides */
    }
}
```

Binary search

Given a sorted array, determine index associated with a given key.

Ex. Dictionary, phone book, book index, ...

Binary search algorithm.
- Examine the middle key.
- If it matches, return its index.
- Otherwise, search either the left or right half.
**Ordered array ST implementation (search)**

```java
public Value get(Key key) {
    int i = bsearch(key);
    if (i == -1) return null;
    return vals[i];
}
```

```java
private int bsearch(Key key) {
    int lo = 0, hi = N-1;
    while (lo <= hi) {
        int m = lo + (hi - lo) / 2;
        int cmp = key.compareTo(keys[m]);
        if (cmp < 0) hi = m - 1;
        else if (cmp > 0) lo = m + 1;
        else if (cmp == 0) return m;
    }
    return -1;
}
```

**Binary search trace**

```
keys[]
[0 1 2 3 4 5 6 7 8 9]
A C E H L M P R S X
```

<table>
<thead>
<tr>
<th>lo</th>
<th>hi</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>A C E H L M P R S X</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>A C E H L M P R S X</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>A C E H L M P R S X</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>A C E H L M P R S X</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>A C E H L M P R S X</td>
</tr>
</tbody>
</table>

entries in black are a[lo..hi]
entry in red is a[m]

loop exits with lo > hi

**Binary search: mathematical analysis**

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

**Def.** \( T(N) = \) number of compares to binary search in a sorted array of size \( N \).

\[
T(N) = T(N/2) + 1
\]

left or right half

**Binary search recurrence.** \( T(N) = T(N/2) + 1 \) for \( N > 1 \), with \( T(1) = 0 \).

- Not quite right for odd \( N \).
- Same recurrence holds for many algorithms.

**Solution.** \( T(N) \sim \lg N \).

- For simplicity, we’ll prove when \( N \) is a power of 2.
- True for all \( N \). [see COS 340]

**Binary search recurrence.** \( T(N) = T(N/2) + 1 \) for \( N > 1 \), with \( T(1) = 0 \).

**Proposition.** If \( N \) is a power of 2, then \( T(N) = \lg N \).

**Pf.**

\[
T(N) = T(N/2) + 1
\]

given

apply recurrence to first term

\[
= T(N/4) + 1 + 1
\]

apply recurrence to first term

\[
= T(N/8) + 1 + 1 + 1
\]

... stop applying, \( T(1) = 0 \)

\[
= T(N/N) + 1 + 1 + ... + 1
\]

\[
= \lg N
\]
Ordered array ST implementation (insert)

Binary search is little help for insert.
• Can find where to insert new key.
• But still need to move larger keys.

```java
public Value put(Key key, Value val) {
    int index = bsearch(key);
    if (index >= 0)
    {
        vals[index] = val;
        return;
    }
    int i = N;
    while (i > 0 && less(key, keys[i]))
    {
        keys[i] = keys[i-1];
        vals[i] = vals[i-1];
        i--;
    }
    vals[i] = val;
    keys[i] = key;
    N++;
}
```

Ordered linked-list ST implementation

Binary search depends on array indexing for efficiency.

Q. How to jump to the middle of a linked list?
A. You can’t do it efficiently.

Ordered list - list advantages.
• Support ordered iterator (for free).
• Cuts search/insert time in half (on average) for random search/insert.
ST implementations: summary

Next 3 lectures. Efficient implementations of search and insert.

<table>
<thead>
<tr>
<th>ST implementation</th>
<th>worst case</th>
<th>average case</th>
<th>ordered iteration?</th>
<th>operations on keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>unordered array</td>
<td>N</td>
<td>N / 2</td>
<td>N</td>
<td>equals()</td>
</tr>
<tr>
<td>unordered list</td>
<td>N</td>
<td>N / 2</td>
<td>N</td>
<td>equals()</td>
</tr>
<tr>
<td>ordered array</td>
<td>log N</td>
<td>log N / 2</td>
<td>yes</td>
<td>compareTo()</td>
</tr>
<tr>
<td>ordered list</td>
<td>N</td>
<td>N / 2</td>
<td>no</td>
<td>compareTo()</td>
</tr>
</tbody>
</table>

ST lookup client

Command line arguments.
• A comma-separated value (CSV) file.
• Key field.
• Value field.

Ex 1. DNS lookup.

```java
public class Lookup {
    public static void main(String[] args) {
        In in = new In(args[0]);
        int keyField = Integer.parseInt(args[1]);
        int valField = Integer.parseInt(args[2]);
        String[] database = in.readAll().split("\n");
        ST<String, String> st = new ST<String, String>();
        for (int i = 0; i < database.length; i++) {
            String[] tokens = database[i].split(",");
            String key = tokens[keyField];
            String val = tokens[valField];
            st.put(key, val);
        }
        while (!StdIn.isEmpty()) {
            String s = StdIn.readString();
            if (!st.contains(s)) StdOut.println("Not found");
            else StdOut.println(st.get(s));
        }
    }
}
```

ST lookup client: Java implementation
Command line arguments.

• A comma-separated value (CSV) file.
• Key field.
• Value field.

Ex 2. Amino acids.

```java
% more amino.csv
TTV, Thr, T, Threonine
TTD, Thr, T, Threonine
TTA, Thr, T, Threonine
TTC, Thr, T, Threonine
TCA, Ser, S, Serine
TCG, Ser, S, Serine
TCT, Ser, S, Serine
TCT, Ser, S, Serine
TAG, Ser, S, Serine
TAT, Tyr, Y, Tyrosine
TAA, Stop, Stop, Stop
TAT, Stop, Stop, Stop
TGG, Trp, W, Tryptophan
```

```
public class SET<Key extends Comparable<Key>>
SET() create an empty set
void add(Key key) add the key to the set
boolean contains(Key key) is the key in the set?
void remove(Key key) remove the key from the set
int size() return the number of keys in the set
Iterator<Key> iterator() iterator through keys in the set
```

Q. How to implement?

Set client example: whitelist

```java
public class Whitelist
{
   public static void main(String[] args)
   {
      SET<String> set = new SET<String>()
      {
         SET<String> set = new SET<String>()
         In in = new In(args[0]);
         while (!in.isEmpty())
            set.add(in.readString());
      
      while (!StdIn.isEmpty())
      {
         String word = StdIn.readString();
         if (set.contains(word))
            StdOut.println(word);
      }
   }
}
```

Q. How to implement?

Set client example: whitelist

```java
public class Whitelist
{
   public static void main(String[] args)
   {
      SET<String> set = new SET<String>()
      {
         SET<String> set = new SET<String>()
         In in = new In(args[0]);
         while (!in.isEmpty())
            set.add(in.readString());
      
      while (!StdIn.isEmpty())
      {
         String word = StdIn.readString();
         if (set.contains(word))
            StdOut.println(word);
      }
   }
}
```
Set client example: blacklist

- Read in a list of words from one file.
- Print out all words from standard input that are not in the list.

```java
public class Blacklist {
    public static void main(String[] args) {
        SET<String> set = new SET<String>();
        In in = new In(args[0]);
        while (!in.isEmpty())
            set.add(in.readString());
        while (!StdIn.isEmpty()) {
            String word = StdIn.readString();
            if (!set.contains(word))
                StdOut.println(word);
        }
    }
}
```

Blacklist and whitelist applications

<table>
<thead>
<tr>
<th>Application</th>
<th>Purpose</th>
<th>Key</th>
<th>In list</th>
</tr>
</thead>
<tbody>
<tr>
<td>spell checker</td>
<td>identify misspelled words</td>
<td>word</td>
<td>dictionary words</td>
</tr>
<tr>
<td>browser</td>
<td>mark visited pages</td>
<td>URL</td>
<td>visited pages</td>
</tr>
<tr>
<td>parental controls</td>
<td>block sites</td>
<td>URL</td>
<td>bad sites</td>
</tr>
<tr>
<td>chess</td>
<td>detect draw</td>
<td>board</td>
<td>positions</td>
</tr>
<tr>
<td>spam filter</td>
<td>eliminate spam</td>
<td>IP address</td>
<td>spam addresses</td>
</tr>
<tr>
<td>credit cards</td>
<td>check for stolen cards</td>
<td>number</td>
<td>stolen cards</td>
</tr>
</tbody>
</table>

Searching challenge 1A

Problem. Maintain symbol table of song names for an iPod.
Assumption A. Hundreds of songs.

Which searching method to use?
1) Unordered array.
2) Ordered linked list.
3) Ordered array with binary search.
4) Need better method, all too slow.
5) Doesn't matter much, all fast enough.
**Searching challenge 1B**

**Problem.** Maintain symbol table of song names for an iPod.

**Assumption B.** Thousands of songs.

Which searching method to use?
1) Unordered array.
2) Ordered linked list.
3) Ordered array with binary search.
4) Need better method, all too slow.
5) Doesn’t matter much, all fast enough.

---

**Searching challenge 2A:**

**Problem.** IP lookups in a web monitoring device.

**Assumption A.** Billions of lookups, millions of distinct addresses.

Which searching method to use?
1) Unordered array.
2) Ordered linked list.
3) Ordered array with binary search.
4) Need better method, all too slow.
5) Doesn’t matter much, all fast enough.

---

**Searching challenge 2B**

**Problem.** IP lookups in a web monitoring device.

**Assumption B.** Billions of lookups, thousands of distinct addresses.

Which searching method to use?
1) Unordered array.
2) Ordered linked list.
3) Ordered array with binary search.
4) Need better method, all too slow.
5) Doesn’t matter much, all fast enough.

---

**Searching challenge 3**

**Problem.** Frequency counts in “Tale of Two Cities.”

**Assumptions.** Book has 135,000+ words; about 10,000 distinct words.

Which searching method to use?
1) Unordered array.
2) Ordered linked list.
3) Ordered array with binary search.
4) Need better method, all too slow.
5) Doesn’t matter much, all fast enough.
Searching challenge 4

Problem. Spell checking for a book.
Assumptions. Dictionary has 25,000 words; book has 100,000+ words.

Which searching method to use?
1) Unordered array.
2) Ordered linked list.
3) Ordered array with binary search.
4) Need better method, all too slow.
5) Doesn't matter much, all fast enough.