Symbol Tables

- API
- sequential search
- binary search
- applications
- challenges

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.

URL	IP address
www.cs.princeton.edu	128.112.136.11
www.princeton.edu	128.112.128.15
www.yale.edu	130.132.143.21
www.harvard.edu	128.103.060.55
www.simpsons.com	209.052.165.60

key value

Symbol table applications

application	purpose of search	key	value
dictionary	look up word	word	definition
book index	find relevant pages	term	list of page numbers
file share	find song to download	name of song	computer ID
financial account	process transactions	account number	transaction details
web search	find relevant web pages	keyword	list of page names
compiler	find properties of variables	variable name	value and type
routing table	route Internet packets	destination	best route
DNS	find IP address given URL	URL	IP address
reverse DNS	find URL given IP address	IP address	URL
genomics	find markers	DNA string	known positions
file system	find file on disk	filename	location on disk

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*	← a[key] = val;
Value	get(Key key)	*return value paired with key*	← a[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()`.

```
public boolean contains(Key key)
{  return get(key) != null;  }
```

- Can implement lazy version of `remove()`.

```
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.

```
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));
}
```

keys	S	E	A	R	C	H	E	X	A	M	P	L	E
values	0	1	2	3	4	5	6	7	8	9	10	11	12

STunordered *output (one possibility)*		ST *output*	
L	11	A	8
P	10	C	4
M	9	E	12
X	7	H	5
H	5	L	9
C	4	M	11
R	3	P	10
A	8	R	3
E	12	S	0
S	0	X	7

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.

- API
- **sequential search**
- binary search
- applications
- challenges

Java conventions for `equals()`

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

Default implementation: `(x == y)`

do `x` and `y` refer to the same object?

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.

```
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);
   }
}
```


Implementing `equals()`

Seems easy, but requires some care.

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

must be `Object`.
Why? Experts still debate.

```
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);
   }
}
```

optimize for true object equality

if I'm executing this code, I'm not `null`

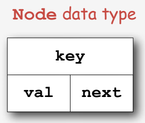
objects must be in the same class

Unordered linked-list ST implementation

Maintain a linked list with keys and values.

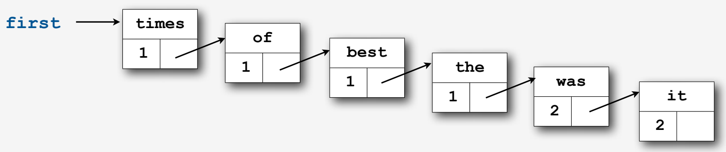
Inner `Node` class.

- 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.



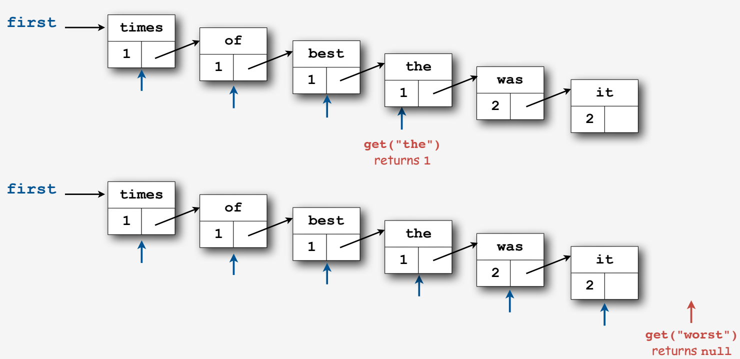
Unordered linked-list ST implementation (skeleton)

```java
public class UnorderedLinkedST<Key, Value>
{
   private Node first;

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

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

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

inner class

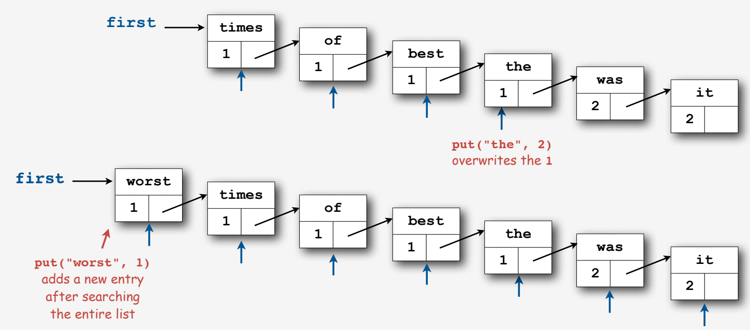
Unordered linked-list ST implementation (search)

```java
public Value get(Key key)
{
   for (Node x = first; x != null; x = x.next))
      if (key.equals(x.key))
         return x.val;
   return null;
}
```

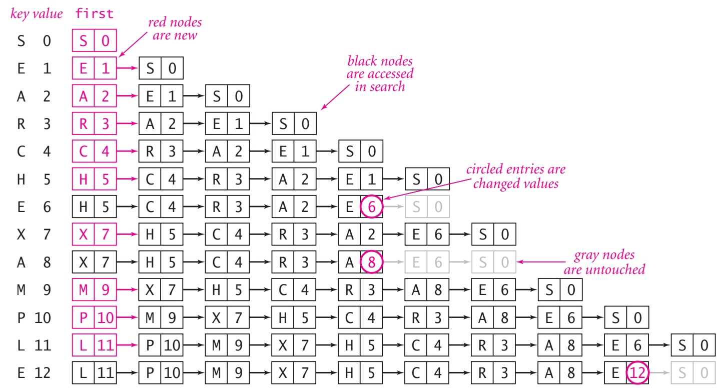


Unordered linked-list ST implementation (insert)

```java
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);
}
```



Unordered linked-list ST: trace



Trace of linked-list ST implementation for standard indexing client

Unordered array ST implementation

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.

N = 6

	0	1	2	3	4	5	6	7
`keys[]`	it	was	the	best	of	times	*null*	*null*
`vals[]`	2	2	1	1	1	1	*null*	*null*

ST implementations: summary

ST implementation	worst case		average case		operations on keys
	search	insert	search hit	insert	
unordered array	N	N	$N/2$	N	`equals()`
unordered list	N	N	$N/2$	N	`equals()`

Challenge. Efficient implementations of search and insert.

Iterators

Goal. Allow client to iterate over the symbol table keys.

```
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;
        }
    }
}
```

Iterable ST client: frequency counter

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

```
% 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
4 the
2 times
4 was
1 wisdom
1 worst
```

tiny example
24 words
10 distinct

```
% more tale.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 FrequencyCount < tale.txt
2941 a
1 aback
1 abandon
10 abandoned
1 abandoning
1 abandonment
1 abashed
1 abate
1 abated
5 abbaye
2 abed
1 abhorrence
1 abided
1 abiding
...
```

real example
137177 words
9888 distinct

Iterable ST client: frequency counter

```
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);
   }
}
```

create ST

read string and update frequency

print all strings

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
4 the
2 times
4 was
1 wisdom
1 worst
```

with one ST implementation

```
% 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
1 foolishness
1 wisdom
2 age
1 worst
2 times
4 of
1 best
4 the
4 was
4 it
```

with `UnorderedLinkedListST`

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

ST implementation	worst case		average case		ordered iteration?	operations on keys
	search	insert	search hit	insert		
unordered array	N	N	$N/2$	N	no	`equals()`
unordered list	N	N	$N/2$	N	no	`equals()`

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

- API
- sequential search
- **binary search**
- applications
- challenges

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.

N = 6

	0	1	2	3	4	5	6	7
`keys[]`	best	it	of	the	times	was	*null*	*null*
`vals[]`	1	2	1	1	1	2	*null*	*null*

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)

```
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)        ← array doubling code omitted
   {
     keys = (Key[])    new Comparable[capacity];   ← standard ugly casts
     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 */  }
}
```

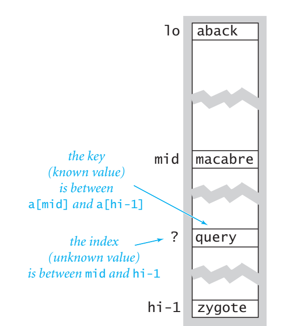

Binary search

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.



Binary search in a sorted array (one step)

Ordered array ST implementation (search)

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

symbol table method

```
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;
}
```

helper binary search method

not found

Binary search trace

keys[]

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

successful search for P

lo	hi	m										
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

entries in black are a[lo..hi]

entry in red is a[m]

unsuccessful search for Q

lo	hi	m										
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 lo > hi

Binary search: mathematical analysis

Proposition. Binary search uses $\sim \lg N$ compares to search any array of size N.

Def. $\mathrm{T}(N) \equiv$ number of compares to binary search in a sorted array of size N.

$$= \underbrace{\mathrm{T}(N/2)}_{\text{left or right half}} \;+\; 1$$

Binary search recurrence. $\mathrm{T}(N) = \mathrm{T}(N/2) + 1$ for $N > 1$, with $\mathrm{T}(1) = 0$.

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

Solution. $\mathrm{T}(N) \sim \lg N$.

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

Binary search recurrence

Binary search recurrence. $\mathrm{T}(N) = \mathrm{T}(N/2) + 1$ for $N > 1$, with $\mathrm{T}(1) = 0$.

Proposition. If N is a power of 2, then $\mathrm{T}(N) = \lg N$.

Pf.

$$\begin{aligned}
\mathrm{T}(N) &= \mathrm{T}(N/2) + 1 && \text{given} \\
&= \mathrm{T}(N/4) + 1 + 1 && \text{apply recurrence to first term} \\
&= \mathrm{T}(N/8) + 1 + 1 + 1 && \text{apply recurrence to first term} \\
&\;\;\cdots \\
&= \mathrm{T}(N/N) + 1 + 1 + \ldots + 1 && \text{stop applying, T(1) = 0} \\
&= \lg N
\end{aligned}$$

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.

	0	1	2	3	4	5	6	7	8	9
keys[]	age	best	it	of	the	times	was	wisdom	*null*	*null*
vals[]	2	1	4	4	4	2	4	1	*null*	*null*

	0	1	2	3	4	5	6	7	8	9
keys[]	age	best	foolish	it	of	the	times	was	wisdom	*null*
vals[]	2	1	1	4	4	4	2	4	1	*null*

`put("foolish", 1)`

Ordered array ST implementation (insert)

```
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++;
}
```

overwrite old value with new value (associative array)

move larger keys over

add the new key-value pair

Ordered array ST implementation: an important special case

Method of choice for some clients.

- Sort database by key.
- Insert N key-value pairs in order by key.
- Support searches that never use more than lg N compares.
- Support occasional (expensive) inserts.

Remark. Takes linear time to insert N keys that are in ascending order if we add the following check to `bsearch()`.

```
int cmp = key.compareTo(keys[N-1]);
if (cmp == 0) return N-1;
if (cmp  > 0) return -1;
```

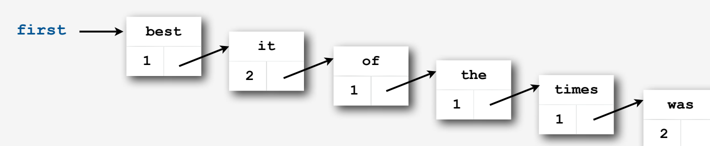

Ordered linked-list ST implementation

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 link-list ST advantages.

- Support ordered iterator (for free).
- Cuts search/insert time in half (on average) for random search/insert.



ST implementations: summary

ST implementation	worst case		average case		ordered iteration?	operations on keys
	search	insert	search hit	insert		
unordered array	N	N	$N/2$	N	no	`equals()`
unordered list	N	N	$N/2$	N	no	`equals()`
ordered array	$\log N$	N	$\log N$	$N/2$	yes	`compareTo()`
ordered list	N	N	$N/2$	$N/2$	yes	`compareTo()`

Next 3 lectures. Efficient implementations of search and insert.

ST lookup client

Command line arguments.

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

Ex 1. DNS lookup.

URL is key IP is value

```
% java Lookup ip.csv 0 1
adobe.com
192.150.18.60
www.princeton.edu
128.112.128.15
ebay.edu
Not found
```

IP is key URL is value

```
% java Lookup ip.csv 1 0
128.112.128.15
www.princeton.edu
999.999.999.99
Not found
```

```
% more ip.csv
www.princeton.edu,128.112.128.15
www.cs.princeton.edu,128.112.136.35
www.math.princeton.edu,128.112.18.11
www.cs.harvard.edu,140.247.50.127
www.harvard.edu,128.103.60.24
www.yale.edu,130.132.51.8
www.econ.yale.edu,128.36.236.74
www.cs.yale.edu,128.36.229.30
espn.com,199.181.135.201
yahoo.com,66.94.234.13
msn.com,207.68.172.246
google.com,64.233.167.99
baidu.com,202.108.22.33
yahoo.co.jp,202.93.91.141
sina.com.cn,202.108.33.32
ebay.com,66.135.192.87
adobe.com,192.150.18.60
163.com,220.181.29.154
passport.net,65.54.179.226
tom.com,61.135.158.237
nate.com,203.226.253.11
cnn.com,64.236.16.20
daum.net,211.115.77.211
blogger.com,66.102.15.100
fastclick.com,205.180.86.4
wikipedia.org,66.230.200.100
rakuten.co.jp,202.72.51.22
...
```


ST lookup client: Java implementation

```
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));
      }
   }
}
```

process input file

build symbol table

process lookups with standard I/O

ST lookup client

Command line arguments.

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

Ex 2. Amino acids.

codon is key   name is value

```
% java Lookup amino.csv 0 3
ACT
Threonine
TAG
Stop
CAT
Histidine
```

```
% more amino.csv
TTT,Phe,F,Phenylalanine
TTC,Phe,F,Phenylalanine
TTA,Leu,L,Leucine
TTG,Leu,L,Leucine
TCT,Ser,S,Serine
TCC,Ser,S,Serine
TCA,Ser,S,Serine
TCG,Ser,S,Serine
TAT,Tyr,Y,Tyrosine
TAC,Tyr,Y,Tyrosine
TAA,Stop,Stop,Stop
TAG,Stop,Stop,Stop
TGT,Cys,C,Cysteine
TGC,Cys,C,Cysteine
TGA,Stop,Stop,Stop
TGG,Trp,W,Tryptophan
CTT,Leu,L,Leucine
CTC,Leu,L,Leucine
CTA,Leu,L,Leucine
CTG,Leu,L,Leucine
CCT,Pro,P,Proline
CCC,Pro,P,Proline
CCA,Pro,P,Proline
CCG,Pro,P,Proline
CAT,His,H,Histidine
CAC,His,H,Histidine
CAA,Gln,Q,Glutamine
CAG,Gln,Q,Glutamine
CGT,Arg,R,Arginine
CGC,Arg,R,Arginine
...
```


ST lookup client

Command line arguments.

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

Ex 3. Class list.

login is key   name is value

```
% java Lookup classlist.csv 3 1
jsh
Jeffrey Scott Harris
dgtwo
Daniel Gopstein
```

login is key   precept is value

```
% java Lookup classlist.csv 3 2
jsh
P01A
```

```
% more classlist.csv
10,Bo Ling,P03,bling
10,Steven A Ross,P01,saross
10,Thomas Oliver Horton Conway,P03,oconway
08,Michael R. Corces Zimmerman,P01A,mcorces
09,Bruce David Halperin,P02,bhalperi
09,Glenn Charles Snyders Jr.,P03,gsnyders
09,Siyu Yang,P01A,siyuyang
08,Taofik O. Kolade,P01,tkolade
09,Katharine Paris Klosterman,P01A,kkloster
SP,Daniel Gopstein,P01,dgtwo
10,Sauhard Sahi,P01,ssahi
10,Eric Daniel Cohen,P01A,edcohen
09,Brian Anthony Geistwhite,P02,bgeistwh
09,Boris Pivtorak,P01A,pivtorak
09,Jonathan Patrick Zebrowski,P01A,jzebrows
09,Dexter James Doyle,P01A,ddoyle
09,Michael Weiyang Ye,P03,ye
08,Delwin Uy Olivan,P02,dolivan
08,Edward George Conbeer,P01A,econbeer
09,Mark Daniel Stefanski,P01,mstefans
09,Carter Adams Cleveland,P03,cclevela
10,Jacob Stephen Lewellen,P02,jlewelle
10,Ilya Trubov,P02,itrubov
09,Kenton William Murray,P03,kwmurray
07,Daniel Steven Marks,P02,dmarks
09,Vittal Kadapakkam,P01,vkadapak
10,Eric Ruben Domb,P01A,edomb
07,Jie Wu,P03,jiewu
08,Pritha Ghosh,P02,prithag
10,Minh Quang Anh Do,P01,mqdo
...
```


Set API

Mathematical set. A collection of distinct keys.

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

- Read in a list of words from one file.
- Print out all words from standard input that are in the list.

```
public class Whitelist
{
   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);
      }
   }
}
```

create empty set of strings

read in whitelist

print strings in list

Set client example: blacklist

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

```
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);
      }
   }
}
```

create empty set of strings

read in blacklist

print strings not in list

Blacklist and whitelist applications

application	purpose	key	in list
spell checker	identify misspelled words	word	dictionary words
browser	mark visited pages	URL	visited pages
parental controls	block sites	URL	bad sites
chess	detect draw	board	positions
spam filter	eliminate spam	IP address	spam addresses
credit cards	check for stolen cards	number	stolen cards

- API
- sequential search
- binary search
- **challenges**

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.