Lecture 10: More on ADT Map

Dr. Kieran T. Herley

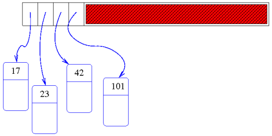
Department of Computer Science
University College Cork

2013/14

Summary

Alternative array-based implementation of ADT Map based on sorted array and (non-recursive) binary search. Use of comparators.

Ordered Array-Based Map

- 

- Ordered Array Representation:
    - array of entries (left-justified) and numEntries as before
    - entries in *increasing order of key*
- Implementation compared to unordered array (UA)
    - get– *binary search*; more efficient than UA
    - put– need to preserve order; less efficient than UA
    - good when get operations predominate

Searching in an Array

Setting Array S of *keys* arranged in nondecreasing order

Problem Given key value x, determine where it appears in S

Simple Solution (Linear Scan) Sweep through S from beginning to end searching for key x

$$\boxed{2}_0\boxed{4}_1\boxed{5}_2\boxed{7}_3\boxed{8}_4\boxed{9}_5\boxed{12}_6\boxed{14}_7$$

Drawbacks Slow for long sequences

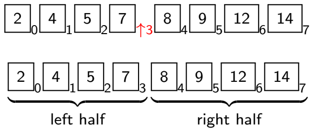
Binary Search- Basic Idea

$$\boxed{2}_{0}\boxed{4}_{1}\boxed{5}_{2}\boxed{7}\uparrow_{3}\boxed{8}_{4}\boxed{9}_{5}\boxed{12}_{6}\boxed{14}_{7}$$

$$\underbrace{\boxed{2}_{0}\boxed{4}_{1}\boxed{5}_{2}\boxed{7}_{3}}_{\text{left half}}\underbrace{\boxed{8}_{4}\boxed{9}_{5}\boxed{12}_{6}\boxed{14}_{7}}_{\text{right half}}$$

- Compare "middle" value against search key
- Outcome ($<$, $=$, $>$) allows us to narrow search to "left half" or "right half" of array
- Idea can be applied iteratively to "home in" on search key
- Technique applies only if array is ordered

Binary Search- Basic Idea cont'd



To search for x in *ordered* array S ($|S| > 1$):

- Let mid denote rank of "midpoint" of S and key(mid) denotes the key value of the item at rank mid of S
- if $x =$ key(mid), then we are done
- if $x <$ key(mid), then x appears in "left half" of S, if at all
- if $x >$ key(mid), then x appears in "right half" of S, if at all

Observation

Task of searching in S simplifies to subtask of searching "half" of S (left "half" or "right" half)

BS Illustration ($k = 22$)

$$\underbrace{\boxed{2}_{0}}_{l}\ \boxed{4}_{1}\ \boxed{5}_{2}\ \boxed{7}_{3}\ \boxed{8}_{4}\ \boxed{9}_{5}\ \boxed{12}_{6}\ \boxed{14}_{7}\ \boxed{17}_{8}\ \boxed{19}_{9}\ \boxed{22}_{10}\ \boxed{25}_{11}\ \boxed{27}_{12}\ \boxed{28}_{13}\ \boxed{33}_{14}\ \underbrace{\boxed{37}_{15}}_{h}$$

BS Illustration ($k = 22$)

$$\underbrace{\boxed{2}_{0}}_{l}\boxed{4}_{1}\boxed{5}_{2}\boxed{7}_{3}\boxed{8}_{4}\boxed{9}_{5}\boxed{12}_{6}\boxed{14}_{7}\boxed{17}_{8}\boxed{19}_{9}\boxed{22}_{10}\boxed{25}_{11}\boxed{27}_{12}\boxed{28}_{13}\boxed{33}_{14}\underbrace{\boxed{37}_{15}}_{h}$$

$$\boxed{2}_{0}\boxed{4}_{1}\boxed{5}_{2}\boxed{7}_{3}\boxed{8}_{4}\boxed{9}_{5}\boxed{12}_{6}\boxed{14}_{7}\underbrace{\boxed{17}_{8}}_{l}\boxed{19}_{9}\boxed{22}_{10}\boxed{25}_{11}\boxed{27}_{12}\boxed{28}_{13}\boxed{33}_{14}\underbrace{\boxed{37}_{15}}_{h}$$

BS Illustration ($k = 22$)

$$\underbrace{\boxed{2}_{0}}_{l}\ \boxed{4}_{1}\ \boxed{5}_{2}\ \boxed{7}_{3}\ \boxed{8}_{4}\ \boxed{9}_{5}\ \boxed{12}_{6}\ \boxed{14}_{7}\ \boxed{17}_{8}\ \boxed{19}_{9}\ \boxed{22}_{10}\ \boxed{25}_{11}\ \boxed{27}_{12}\ \boxed{28}_{13}\ \boxed{33}_{14}\ \underbrace{\boxed{37}_{15}}_{h}$$

$$\boxed{2}_{0}\ \boxed{4}_{1}\ \boxed{5}_{2}\ \boxed{7}_{3}\ \boxed{8}_{4}\ \boxed{9}_{5}\ \boxed{12}_{6}\ \boxed{14}_{7}\ \underbrace{\boxed{17}_{8}}_{l}\ \boxed{19}_{9}\ \boxed{22}_{10}\ \boxed{25}_{11}\ \boxed{27}_{12}\ \boxed{28}_{13}\ \boxed{33}_{14}\ \underbrace{\boxed{37}_{15}}_{h}$$

$$\boxed{2}_{0}\ \boxed{4}_{1}\ \boxed{5}_{2}\ \boxed{7}_{3}\ \boxed{8}_{4}\ \boxed{9}_{5}\ \boxed{12}_{6}\ \boxed{14}_{7}\ \underbrace{\boxed{17}_{8}}_{l}\ \boxed{19}_{9}\ \boxed{22}_{10}\ \underbrace{\boxed{25}_{11}}_{h}\ \boxed{27}_{12}\ \boxed{28}_{13}\ \boxed{33}_{14}\ \boxed{37}_{15}$$

BS Illustration ($k = 22$)

$$\underbrace{\boxed{2}_{0}}_{l}\ \boxed{4}_{1}\ \boxed{5}_{2}\ \boxed{7}_{3}\ \boxed{8}_{4}\ \boxed{9}_{5}\ \boxed{12}_{6}\ \boxed{14}_{7}\ \boxed{17}_{8}\ \boxed{19}_{9}\ \boxed{22}_{10}\ \boxed{25}_{11}\ \boxed{27}_{12}\ \boxed{28}_{13}\ \boxed{33}_{14}\ \underbrace{\boxed{37}_{15}}_{h}$$

$$\boxed{2}_{0}\ \boxed{4}_{1}\ \boxed{5}_{2}\ \boxed{7}_{3}\ \boxed{8}_{4}\ \boxed{9}_{5}\ \boxed{12}_{6}\ \boxed{14}_{7}\ \underbrace{\boxed{17}_{8}}_{l}\ \boxed{19}_{9}\ \boxed{22}_{10}\ \boxed{25}_{11}\ \boxed{27}_{12}\ \boxed{28}_{13}\ \boxed{33}_{14}\ \underbrace{\boxed{37}_{15}}_{h}$$

$$\boxed{2}_{0}\ \boxed{4}_{1}\ \boxed{5}_{2}\ \boxed{7}_{3}\ \boxed{8}_{4}\ \boxed{9}_{5}\ \boxed{12}_{6}\ \boxed{14}_{7}\ \underbrace{\boxed{17}_{8}}_{l}\ \boxed{19}_{9}\ \boxed{22}_{10}\ \underbrace{\boxed{25}_{11}}_{h}\ \boxed{27}_{12}\ \boxed{28}_{13}\ \boxed{33}_{14}\ \boxed{37}_{15}$$

$$\boxed{2}_{0}\ \boxed{4}_{1}\ \boxed{5}_{2}\ \boxed{7}_{3}\ \boxed{8}_{4}\ \boxed{9}_{5}\ \boxed{12}_{6}\ \boxed{14}_{7}\ \boxed{17}_{8}\ \boxed{19}_{9}\ \underbrace{\boxed{22}_{10}}_{l}\ \underbrace{\boxed{25}_{11}}_{h}\ \boxed{27}_{12}\ \boxed{28}_{13}\ \boxed{33}_{14}\ \boxed{37}_{15}$$

BS Illustration ($k = 22$)

$$\underbrace{\boxed{2}_0}_{l}\boxed{4}_1\boxed{5}_2\boxed{7}_3\boxed{8}_4\boxed{9}_5\boxed{12}_6\boxed{14}_7\boxed{17}_8\boxed{19}_9\boxed{22}_{10}\boxed{25}_{11}\boxed{27}_{12}\boxed{28}_{13}\boxed{33}_{14}\underbrace{\boxed{37}_{15}}_{h}$$

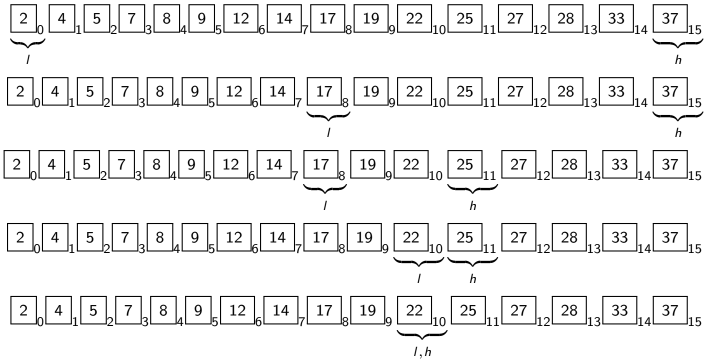
$$\boxed{2}_0\boxed{4}_1\boxed{5}_2\boxed{7}_3\boxed{8}_4\boxed{9}_5\boxed{12}_6\boxed{14}_7\underbrace{\boxed{17}_8}_{l}\boxed{19}_9\boxed{22}_{10}\boxed{25}_{11}\boxed{27}_{12}\boxed{28}_{13}\boxed{33}_{14}\underbrace{\boxed{37}_{15}}_{h}$$

$$\boxed{2}_0\boxed{4}_1\boxed{5}_2\boxed{7}_3\boxed{8}_4\boxed{9}_5\boxed{12}_6\boxed{14}_7\underbrace{\boxed{17}_8}_{l}\boxed{19}_9\boxed{22}_{10}\underbrace{\boxed{25}_{11}}_{h}\boxed{27}_{12}\boxed{28}_{13}\boxed{33}_{14}\boxed{37}_{15}$$

$$\boxed{2}_0\boxed{4}_1\boxed{5}_2\boxed{7}_3\boxed{8}_4\boxed{9}_5\boxed{12}_6\boxed{14}_7\boxed{17}_8\boxed{19}_9\underbrace{\boxed{22}_{10}}_{l}\underbrace{\boxed{25}_{11}}_{h}\boxed{27}_{12}\boxed{28}_{13}\boxed{33}_{14}\boxed{37}_{15}$$

$$\boxed{2}_0\boxed{4}_1\boxed{5}_2\boxed{7}_3\boxed{8}_4\boxed{9}_5\boxed{12}_6\boxed{14}_7\boxed{17}_8\boxed{19}_9\underbrace{\boxed{22}_{10}}_{l,h}\boxed{25}_{11}\boxed{27}_{12}\boxed{28}_{13}\boxed{33}_{14}\boxed{37}_{15}$$

BS Illustration ($k = 22$)



Variables ℓ and h to delimit "search interval"; "search interval" halves at each stage; ultimately converges on search item (if present)

Binary Search

```
Algorithm BinarySearch(S, k)
  low ← 0
  high ← S.size() − 1
  while low ≤ high do
    mid = (low + high)/2
    midKey = key(mid)
    if k = midKey then
      return mid
    else
    if k < midKey then
      high ← mid − 1
    else
      low ← mid + 1
  return −1
```

Behaviour Returns index of slot within S that houses search key k (or -1 if there is no such index)

Binary Search

```
Algorithm BinarySearch(S, k)
    low $\leftarrow$0
    high $\leftarrow$S.size() $-$ 1
    while low $\leq$high do
        mid = (low + high)/2
        midKey = key(mid)
        if k = midKey then
            return mid
        else
        if k < midKey then
            high $\leftarrow$mid $-$1
        else
            low $\leftarrow$mid+1
    return $-1$
```

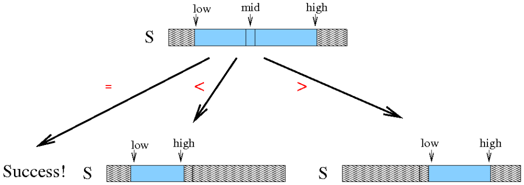
Behaviour Returns index of slot within S that houses search key k (or -1 if there is no such index)

Notes

- mid = (low + high)/2 is *index* midway between low and high.
- key(mid) denotes *key* housed at that index
- low, high updated according to comparison between key(mid) and k to narrow search interval at each iteration

Aside – Recursive BS

```
Algorithm BinarySearch(S, k, low, high)
    if low > high then
        return −1
    else
        mid ←(low + high)/2
        midKey ←key(mid)
        if k = midKey then
            return mid
        else
        if k < midKey then
            return
              BinarySearch(S,k, low, m
        else
            return
              BinarySearch(S,k,
mid+1, high)
```



To be discussed later

BS in Action ($k = 22$– present)

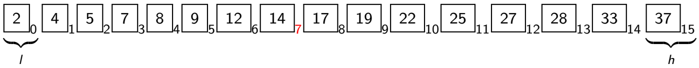
```
Algorithm BinarySearch(S, k)
    low ←0
    high ←S.size() − 1
    while low ≤high do
        mid = (low + high)/2
        midKey = key(mid)
        if k = midKey then
            return mid
        else
        if k < midKey then
            high ←mid −1
        else
            low ←mid+1
    return −1
```

Observation

Trace slightly different from earlier illustration since k=midKey case terminates algorithm

Observation

Returns 10 *i.e. index of slot housing search key*



BS in Action ($k = 22$– present)

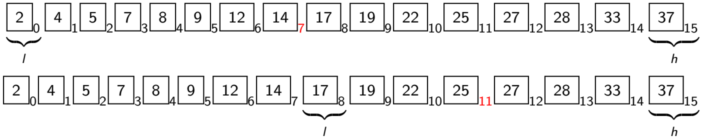
```
Algorithm BinarySearch(S, k)
  low ←0
  high ←S.size() − 1
  while low ≤high do
    mid = (low + high)/2
    midKey = key(mid)
    if k = midKey then
      return mid
    else
    if k < midKey then
      high ←mid −1
    else
      low ←mid+1
  return −1
```

Observation

Trace slightly different from earlier illustration since k=midKey case terminates algorithm

Observation

Returns 10 *i.e. index of slot housing search key*



BS in Action ($k = 22$– present)

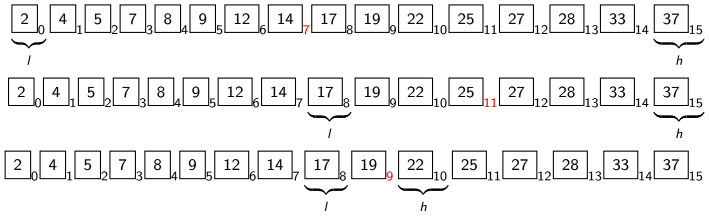
```
Algorithm BinarySearch(S, k)
    low ←0
    high ←S.size() − 1
    while low ≤high do
        mid = (low + high)/2
        midKey = key(mid)
        if k = midKey then
            return mid
        else
        if k < midKey then
            high ←mid −1
        else
            low ←mid+1
    return −1
```

Observation

Trace slightly different from earlier illustration since k=midKey case terminates algorithm

Observation

Returns 10 *i.e. index of slot housing search key*



BS in Action ($k = 22$– present)

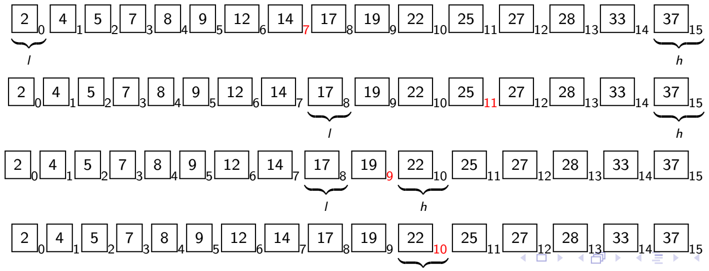
```
Algorithm BinarySearch(S, k)
    low ←0
    high ←S.size() − 1
    while low ≤high do
        mid = (low + high)/2
        midKey = key(mid)
        if k = midKey then
            return mid
        else
        if k < midKey then
            high ←mid −1
        else
            low ←mid+1
    return −1
```

Observation

Trace slightly different from earlier illustration since k=midKey case terminates algorithm

Observation

Returns 10 *i.e. index of slot housing search key*

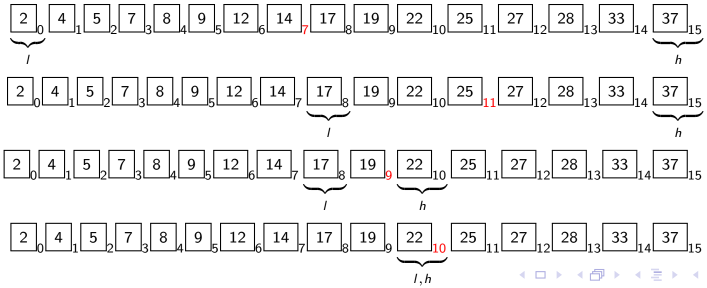


BS in Action ($k = 23$– not present)

```
Algorithm BinarySearch(S, k)
    low ←0
    high ←S.size() − 1
    while low ≤high do
        mid = (low + high)/2
        midKey = key(mid)
        if k = midKey then
            return mid
        else
        if k < midKey then
            high ←mid −1
        else
            low ←mid+1
    return −1
```

Observation

Returns −1, signifying search key not found

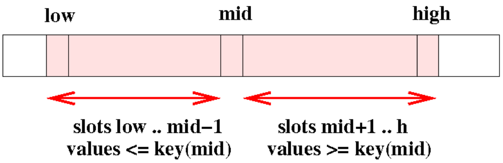


Why Does BS Work?

```
Algorithm BinarySearch(S, k)
    low ←0
    high ←S.size() − 1
    while low ≤high do
        Invariant
        mid = (low + high)/2
        midKey = key(mid)
        if k = midKey then
            return mid
        else
        if k < midKey then
            high ←mid −1
        else
            low ←mid+1
    return −1
```

Claim

The following invariant is true at beginning of every execution of loop body: "If present, k lies within $S[low..high]$"

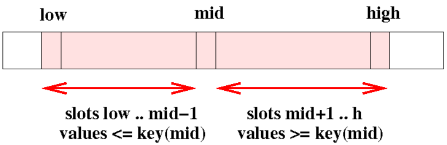


Why? cont'd

```
Algorithm BinarySearch(S, k)
    low ← 0
    high ← S.size() − 1
    while low ≤ high do
        Invariant
        mid = (low + high)/2
        midKey = key(mid)
        if k = midKey then
            return mid
        else
        if k < midKey then
            high ← mid − 1
        else
            low ← mid + 1
    return −1
```

- Suppose Invariant true at begining of loop body (with low $= \ell$ and high $= h$)
- Effect of loop body execution

condition	end low	end high
$k = \text{key}(\text{mid})$	ℓ	h
$k < \text{key}(\text{mid})$	ℓ	mid − 1
$k > \text{key}(\text{mid})$	mid + 1	h



- Invariant also true *after* loop body execution (for new values of low and high)

Why? cont'd

```
Algorithm BinarySearch(S, k)
    low $\leftarrow 0$
    high $\leftarrow$S.size() $- 1$
    while low $\leq$ high do
        Invariant
        mid $=$ (low $+$ high)/2
        midKey $=$ key(mid)
        if k $=$ midKey then
            return mid
        else
        if k $<$ midKey then
            high $\leftarrow$mid $-1$
        else
            low $\leftarrow$mid$+1$
    return $-1$
```

Note: When search key not present, algorithm can only exit at return -1 line

Halving Property

Assumption array length a power of two

Claim Each iteration halves the search interval length

Why?

- $S[\ell..\text{mid}-1]$:

$$\underbrace{(\text{mid}-1)-\ell+1}_{\text{length of } S[\ell..\text{mid}-1]} = (\ell+h)/2-\ell \leq \underbrace{(h-\ell+1)/2}_{\text{half length of } S[\ell..h]}$$

(Recall that / denotes *integer* division)

- $S[\text{mid}+1..h]$: similar reasoning

Efficiency of Binary Search

- Each iteration halves the search interval length
stage	search interval
0	n
1	$n/2$
2	$n/4$
$\dots$	$\dots$
i	$n/2^i$
$\dots$	$\dots$
$\log_2 n$	1

Observation

There are $1 + \log_2 n$ iterations at most

- The overall "running time" is proportional to $\log n$ rather than n as in the case of linear search and hence more efficient.

A Problem With Comparisons

- So far implementation relies on .equals() for comparions– restrictive
  - Not all classes support .equals() in a natural way
  - Might want to "redefine" notion of equality/order among existing types: *e.g.* ignore minor spelling/capitalization variations:
    - "O'Sullivan" = "O Sullivan"
    - "FitzGerald" = "Fitzgerald"
  - How to express $<$ or $>$ for binary search?
- Syntactic packaging differs for comparisons involving different Java type
  - Strings use compareTo
  - Integers use intValue
  - *etc.*
- How to express comparisons within ArrayBasedMap in type-independent but flexible way?

Comparators

- Intuitively a comparator is a comparing device (object): it allows us to compare objects of some particlar type
-
```
Comparator comp = . . .
    . . .
if (comp.compare(a, b) < 0)
{ System.out. println ("a␣is␣smaller");}
```
- Note: allows comparison to be expressed without tying to any specific type for *a* and *b*

ADT Comparator

- Comparator's compare operation

  compare(a, b)**:** Return an integer i such that $i < 0$ if $a < b$, $i = 0$ if $a = b$ and $i > 0$ if $a > b$. Illegal if a and b cannot be compared. *Input: KeyType, KeyType*; *Output: int*.

-

```
public interface Comparator<KeyType>
{  public int compare(KeyType a, KeyType b);
}
```

Integer Comparator Implementation

```
public class IntegerComparator
       implements Comparator<Integer>
{
  public int compare(Integer a, Integer b)
   {  checkIfComparable(a); checkIfComparable(b);
      int aValue = a.intValue();
      int bValue = b.intValue();
      return (aValue − bValue);
   }
   // OTHER METHODS
}
```

String Comparator Implementation

```
public class StringComparator
      implements Comparator<String>
{
   public int compare(String a, String b)
   {  checkIfComparable(a); checkIfComparable(b);
      int compResult = a.compareTo( b);
      return compResult ;
   }

   // OTHER METHODS
}
```

ArrayBasedMap

```
public  class  ArrayBasedMap2<KeyType, ValueType>
               extends ArrayBasedMap<KeyType, ValueType>
               implements Map<KeyType, ValueType>
{   public  ArrayBasedMap2(Comparator<KeyType> c)
    {   . . .
        comp = c;
    }
    . . .
    protected Comparator<KeyType> comp;
}
```

- ArrayBasedMap2 has comparator instance variable named comp
- Comparator supplied (as argument to constructor) when map created (next slide)
- Comparisons within ArrayBasedMap2 rewritten to use binary search use comp for comparisons
- Inherits instance vars, method get and remove; overrides findEntry and put

Creating Different Types of Map

Integer keys

```
Map<Integer, SomeType> myMap1 =
    new ArrayBasedMap2<Integer, SomeType>(new IntegerComparator())
```

String keys

```
Map<String, SomeType> myMap2 =
    new ArrayBasedMap2<String, SomeType>(new StringComparator());
```

ArrayBasedMap2

```
public class ArrayBasedMap2<KeyType, ValueType>
             extends ArrayBasedMap<KeyType, ValueType>
             implements Map<KeyType, ValueType>
{  public ArrayBasedMap2(Comparator<KeyType> c)
   {  . . .
      comp = c;
   }
   . . .
   protected Comparator<KeyType> comp;
}
```

ArrayBasedMap2 cont'd

```
private int findEntry(KeyType key)
{  int low = 0;
   int high = this.size()-1;
   while (low <= high)
   {  int mid = (low + high)/2;

      Entry<KeyType, ValueType> e = entries[mid];
      int compResult = comp.compare(key, e.getKey());
      if (compResult == 0)
      {  return mid;
      }
      else
      if (compResult < 0)
      {  high = mid -1;
      }
      else
      {  low = mid + 1;
      }
   }
   return NO_SUCH_KEY;
```

- Overrides linear-search findEntry of ArrayBasedMap
- NB: Use of comp for comparisons

A Note on Other Methods

Objective	Must preserve order across insertions and deletions
get/remove	Inherited from ArrayBasedMap
put	

- use findEntry to check presence/location
- if absent,
    - (determine suitable insertion point)
    - shift entries rightwards to create gap
    - place new entry in gap
- if present
    - replace old value with new

Simple Spelling Checker

Goal A Java application that reads a document and flags the misspelt words.

Idea

- Maintain list of common words
- Scan through document
  - (ignore non-“words”–spaces, punctuation)
  - lookup each “word” and if not on list flag it as possible misspelling

Challenges

- document reading apparatus
- word list apparatus

WordReader

- Simple document-reading class for text files
- WordReader r = **new** WordReader("myDoc.txt");
- Main word-reader operation:

  nextWord(): Return the next word (in lowercase), if any. Return null if no words remain. *Input: None*; *Output: String*.

  - Any maximal sequence of letters constitutes a word. All non-letters (spaces, punctuation *etc.*) are treated as inter-word space and ignored.
- Implementation not considered here. See code on webpage for details.

WordList

- Simple word list abstraction based on “commonest” English language words.
- Creating a word list:

```
WordList legalWords = new WordList();
```

- Main word list operation:

  isWord(str): Return true if str appears in word list and false otherwise. *Input: String*; *Output: boolean*.

SpellingChecker

```
WordList legalWords = new WordList();
WordReader document =
   new WordReader(/* name of document */);
String wordFromDoc = document.nextWord();
while (wordFromDoc != null)
{   if (! legalWords.isWord(wordFromDoc))
    {   System.out.println(
            "spelling␣error␣"+ wordFromDoc+"␣?");
    }
    wordFromDoc = document.nextWord();
}
```

WordList

```
public class WordList
{  public WordList()
   {   initializeWordList ( wordListFile );}
   public boolean isWord(String str){  /* next slide */ }
   private void  initializeWordList (String fileName)
   {  /* read content of file and store in wordList */}
   private static final String wordListFile =
      "commonest2000.txt";
   private Map<String, String> wordList;
}
```

initializeWordList Issues

- Source of words
- Reading Words from Source
- Storing the Words

initializeWordList Code

```
private void initializeWordList (
    String wordListFile )
{   wordList =
      new ArrayBasedMap<String, String>(new StringComparator());
    WordReader reader =
        new WordReader(wordListFile);
    String sourceWord;
    sourceWord = reader.nextWord();
    while (sourceWord != null)
    {   wordList.put(sourceWord, sourceWord);
        sourceWord = reader.nextWord();
    }
}
```

isWord Code

```
public boolean isWord(String str)
{   String   wordEntry = wordList.get( str );
    return (wordEntry != null);
}
```

Something To Think About

- Quality of word-list determines effectiveness of spelling-checker
- How would you assemble such a list?

Translations

Problem Implement a simple utility for translating documents from English into German.

Simplification Use word-by-word strategy: "translate" each English word into its German "equivalent"; English-German equivalence dictated by English-German map.

Idea

- Use ADT Map D to store English-German pairs for large collection of common words
- Process English document word by word:
    - For each word,
    - look word up in D
    - output its German equivalent

Note Yields awful translations; can provide starting point for more powerful techniques though.