4.4 Symbol Tables

Symbol Table

Symbol table. Key-value pair abstraction.
- Insert a key with specified value.
- 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 API

public class ST<Key key, Val val>		(symbol table data type)
	ST()	create an empty symbol table
void	put(Key key, Val val)	insert a key-value pair
Val	get(Key key)	return value associated with given key
boolean	contains(Key key)	is the given key present?
void	remove(Key key)	delete the key and associated value
Iterator<Key>	iterator()	return an iterator over the keys

```
public static void main(String[] args) {
   ST<String, String> st = new ST<String, String>();

   st.put("www.cs.princeton.edu", "128.112.136.11");
   st.put("www.princeton.edu",    "128.112.128.15");
   st.put("www.yale.edu",         "130.132.143.21");
      st["www.yale.com"] = "209.052.165.60"

   StdOut.println(st.get("www.cs.princeton.edu"));
   StdOut.println(st.get("www.harvardsucks.com"));
   StdOut.println(st.get("www.yale.com"));
}
                       st["www.yale.edu"]
```

```
128.112.136.11
null
130.132.143.21
```


Symbol Table Applications

Application	Purpose	Key	Value
Phone book	Look up phone number	Name	Phone number
Bank	Process transaction	Account number	Transaction details
File share	Find song to download	Name of song	Computer ID
File system	Find file on disk	Filename	Location on disk
Dictionary	Look up word	Word	Definition
Web search	Find relevant documents	Keyword	List of documents
Book index	Find relevant pages	Keyword	List of pages
Web cache	Download	Filename	File contents
Genomics	Find markers	DNA string	Known positions
DNS	Find IP address given URL	URL	IP address
Reverse DNS	Find URL given IP address	IP address	URL
Compiler	Find properties of variable	Variable name	Value and type
Routing table	Route Internet packets	Destination	Best route

Symbol Table Client: Frequency Counter

Frequency counter. [e.g., web traffic analysis, linguistic analysis]

- Read in a key.
- If key is in symbol table, increment counter by one;
  If key is not in symbol table, insert it with count = 1.

```
public class FrequencyCounter {
   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, st.get(key) + 1);
         else                  st.put(key, 1);
      }                                          calculate frequencies

      // enhanced for loop
      for (String s : st)
         StdOut.println(st.get(s) + " " + s);    print results
   }
}
```


Datasets

Linguistic analysis. Compute word frequencies in a corpus of text.

File	Description	Words	Distinct
mobydict.txt	Melville's Moby Dick	210,028	16,834
leipzig100k.txt	100K random sentences	2,121,054	144,256
leipzig200k.txt	200K random sentences	4,238,435	215,515
leipzig1m.txt	1M random sentences	21,191,455	534,580

Reference: Wortschatz corpus, Universität Leipzig
http://corpora.informatik.uni-leipzig.de

Zipf's Law

Linguistic analysis. Compute word frequencies in a piece of text.

```
% java Freq < mobydick.txt
4583 a
2 aback
2 abaft
3 abandon
7 abandoned
1 abandonedly
2 abandonment
2 abased
1 abasement
2 abashed
1 abate
…
```

```
% java Freq < mobydick.txt | sort -rn
13967 the
6415 of
6247 and
4583 a
4508 to
4037 in
2911 that
2481 his
2370 it
1940 i
1793 but
…
```

Zipf's law. In natural language, frequency of i^{th} most common word is proportional to i.

e.g., most frequent word occurs about twice as often as second most frequent one

Zipf's Law

Linguistic analysis. Compute word frequencies in a piece of text.

```
% java Freq < leipzig1m.txt | sort -rn
1160105 the
593492 of
560945 to
472819 a
435866 and
430484 in
205531 for
192296 The
188971 that
172225 is
148915 said
…
```

Zipf's law. In natural language, frequency of i^{th} most common word is proportional to i.

e.g., most frequent word occurs about twice as often as second most frequent one

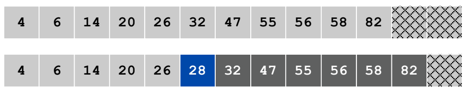
Symbol Table: Brute Force Implementations

Unsorted array.

- Put: add key to the end (if not already there).
- Get: scan through all keys to find desired value.

Sorted array.

- Put: find insertion point, and shift all larger keys right.
- Get: binary search to find desired key.



insert 28

Symbol Table: Implementations Cost Summary

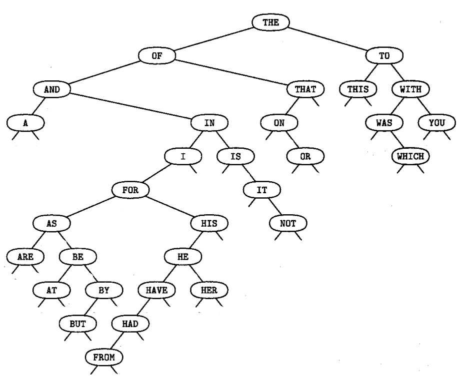
Unordered array. Hopelessly slow for large inputs.

Ordered array. Acceptable if many more searches than inserts; too slow if many inserts.

	Running Time		Frequency Count			
Implementation	Search	Insert	Moby	100K	200K	1M
Unordered array	N	N	170 sec	4.1 hr	-	-
Ordered array	log N	N	5.8 sec	5.8 min	15 min	2.1 hr

Challenge. Make all ops logarithmic.

Binary Search Trees



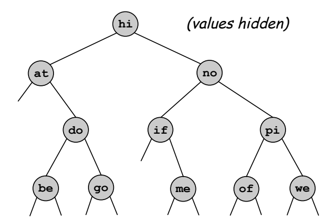
Reference: Knuth, The Art of Computer Programming

Binary Search Trees

Def. A binary search tree is a binary tree in symmetric order.

Binary tree is either:

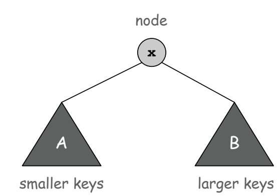
- Empty.
- A key-value pair and two binary trees.



(values hidden)

Symmetric order.

- Keys in left subtree are smaller than parent.
- Keys in right subtree are larger than parent.



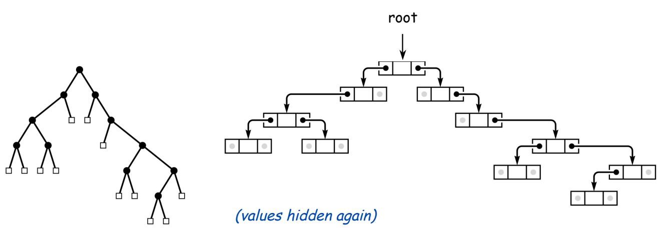
Binary Tree: Java Implementation

To implement: use two links per **Node**.

A **Node** is comprised of:

- A key.
- A value
- A reference to the left subtree.
- A reference to the right subtree.

```
private class Node {
   private Key key;
   private Val val;
   private Node left;
   private Node right;
}
```



(values hidden again)

BST: Skeleton

BST. Allow generic keys and values.

allows `String` and `Integer` keys; see book for details

```
public class BST<Key extends Comparable, Val> {
   private Node root;   // root of the BST

   private class Node {
      private Key key;
      private Val val;
      private Node left, right;

      private Node(Key key, Val val) {
         this.key = key;
         this.val = val;
      }
   }

   public void put(Key key, Val val)  { … }
   public Val get(Key key)            { … }
   public boolean contains(Key key)   { … }

}
```


BST: Search

Get. Return `val` corresponding to given `key`, or `null` if no such key.

```
public Val get(Key key) {
   return get(root, key);
}

private Val get(Node x, Key key) {
   if (x == null) return null;
   int cmp = key.compareTo(x.key);
   if      (cmp < 0) return get(x.left,  key);
   else if (cmp > 0) return get(x.right, key);
   else              return x.val;
}

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

negative if less, zero if equal, positive if greater

BST: Insert

Put. Associate `val` with `key`.

- Search, then insert.
- Concise (but tricky) recursive code.

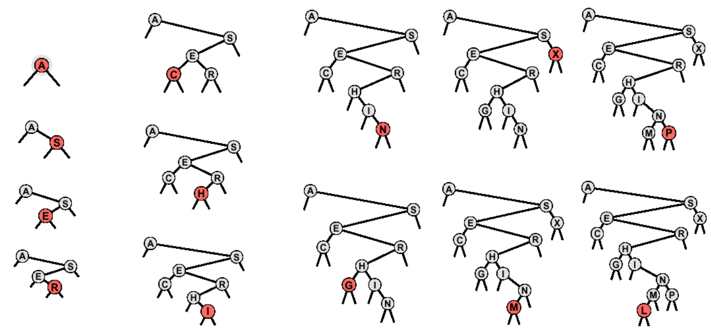
```
public void put(Key key, Val val) {
   root = insert(root, key, val);
}

private Node insert(Node x, Key key, Val val) {
   if (x == null) return new Node(key, val);
   int cmp = key.compareTo(x.key);
   if      (cmp < 0) x.left  = insert(x.left,  key, val);
   else if (cmp > 0) x.right = insert(x.right, key, val);
   else x.val = val;
   return x;
}
```

overwrite old value with new value

BST Insertion Example

BST insert. **A S E R C H I N G X M P L**



BST Implementation: Practice

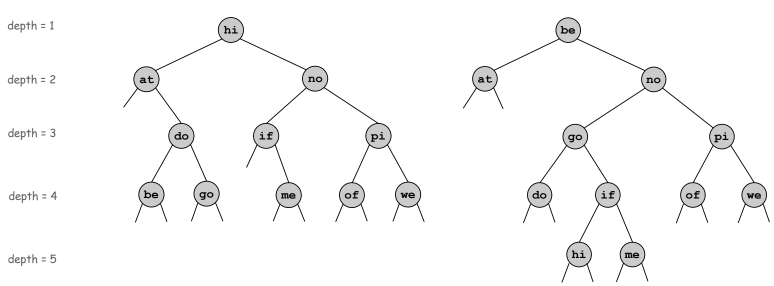
Bottom line. Difference between a practical solution and no solution.

	Running Time		Frequency Count			
Implementation	Search	Insert	Moby	100K	200K	1M
Unordered array	N	N	170 sec	4.1 hr	-	-
Ordered array	log N	N	5.8 sec	5.8 min	15 min	2.1 hr
BST	?	?	.95 sec	7.1 sec	14 sec	69 sec

BST: Analysis

Running time per put/get.
- There are many BSTs that correspond to same set of keys.
- Cost is proportional to depth of node.

number of nodes on path from root to node

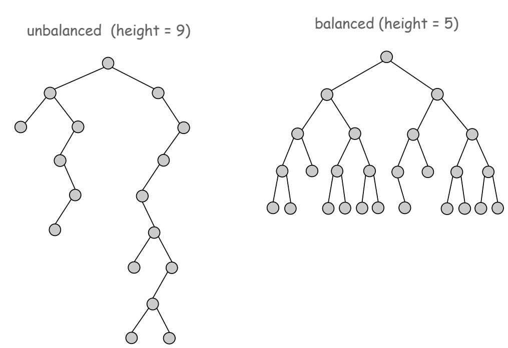


BST: Analysis

Fact. If keys are inserted in random order, average depth ≈ 2N ln N and expected height ≈ 4.31 ln N.

maximum depth

Corollary. Search and insert take logarithmic time on average.



Symbol Table: Implementations Cost Summary

BST. Logarithmic time ops if keys inserted in random order.

	Running Time		Frequency Count			
Implementation	Search	Insert	Moby	100K	200K	1M
Unordered array	N	N	170 sec	4.1 hr	-	-
Ordered array	log N	N	5.8 sec	5.8 min	15 min	2.1 hr
BST	log N †	log N †	.95 sec	7.1 sec	14 sec	69 sec

† assumes keys inserted in random order

Q. Can we guarantee logarithmic performance?

Red-Black Tree

Red-black tree. A clever BST variant that guarantees height ≤ 2 lg N.

↖ see COS 226

	Running Time		Frequency Count			
Implementation	Search	Insert	Moby	100K	200K	1M
Unordered array	N	N	170 sec	4.1 hr	-	-
Ordered array	log N	N	5.8 sec	5.8 min	15 min	2.1 hr
BST	log N †	log N †	.95 sec	7.1 sec	14 sec	69 sec
Red-black	log N	log N	.95 sec	7.0 sec	14 sec	74 sec

† assumes keys inserted in random order

Symbol Table: Summary

Symbol table. Quintessential database lookup data type.

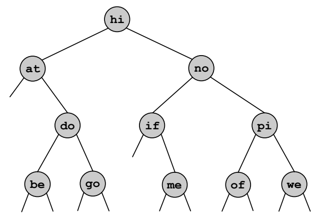
Choices. Ordered array, unordered array, BST, red-black, hash, ….
- Different implementations have different performance characteristics.
- Java libraries: `TreeMap`, `HashMap`.

Remark. Better symbol table implementation improves all clients.

Iteration

Inorder Traversal

Inorder traversal.
- Recursively visit left subtree.
- Visit node.
- Recursively visit right subtree.



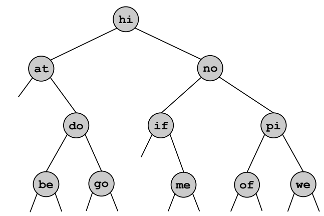
inorder: at be do go hi if me no of pi we

```
public inorder() { inorder(root); }

private void inorder(Node x) {
   if (x == null) return;
   inorder(x.left);
   StdOut.println(x.key);
   inorder(x.right);
}
```


Preorder Traversal

Preorder traversal.
- Visit node.
- Recursively visit left subtree.
- Recursively visit right subtree.



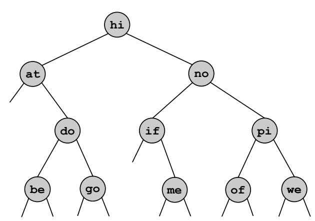
preorder: hi at do be go no if me pi of we

```
public preorder() { preorder(root); }

private void preorder(Node x) {
   if (x == null) return;
   preorder(x.left);
   StdOut.println(x.key);
   preorder(x.right);
}
```


Postorder Traversal

Postorder traversal.
- Recursively visit left subtree.
- Recursively visit right subtree.
- Visit node.



postorder: be go do at me if of we pi no hi

```
public postorder() { postorder(root); }

private void postorder(Node x) {
   if (x == null) return;
   postorder(x.left);
   postorder(x.right);
   StdOut.println(x.key);
}
```


Enhanced For Loop

Enhanced for loop. Enable client to iterate over items in a collection.

```
ST<String, Integer> st = new ST<String, Integer>();
…
for (String s : st) {
   StdOut.println(st.get(s) + " " + s);
}
```

Enhanced For Loop with BST

BST. Add following code to make compatible with enhanced for loop.

see COS 226 for details

```
import java.util.Iterator;
import java.util.NoSuchElementException;

public class BST<Key extends Comparable, Val> implements Iterable<Key> {
   private Node root;

   private class Node { … }

   public void put(Key key, Val val) { … }
   public Val get(Key key)            { … }
   public boolean contains(Key key)   { … }

   public Iterator<Key> iterator() { return new Inorder(); }
      private class Inorder implements Iterator<Key> {
         private Stack<Node> stack = new Stack<Node>();
         Inorder() {
            Node x = root;
            while (x != null) { stack.push(x); x = x.left; }
      }
      public boolean hasNext()  { return !stack.isEmpty(); }
      public Key next() {
         if (!hasNext()) throw new NoSuchElementException();
         Node x = stack.pop();
         Key key = x.key;
         x = x.right;
         while (x != null) { stack.push(x); x = x.left; }
         return key;
      }
   }
}
```


Inverted Index

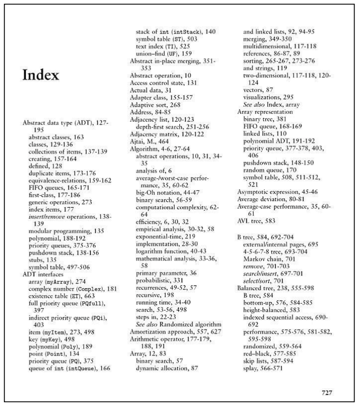
Inverted Index

Inverted index. Given a list of pages, preprocess them so that you can quickly find all pages containing a given query word.

Ex 1. Book index.
Ex 2. Web search engine index.
Ex 3. File index (e.g, Spotlight).

Symbol table.
- Key = query word.
- Value = set of pages.

no duplicates


Index

727

Set API

Set. Unordered collection of distinct keys.

`public class SET<Key key>`		(set data type)
	`SET()`	create an empty set
`void`	`add(Key key)`	add a key to the set
`boolean`	`contains(Key key)`	is the given key in the set?
`void`	`remove(Key key)`	delete the key from the set
`Iterator<Key>`	`iterator()`	return an iterator over the keys

Efficient implementation. Same as symbol table, but ignore value.

Inverted Index: Java Implementation

```
public class InvertedIndex {
   public static void main(String[] args) {
      ST<String, SET<String>> st = new ST<String, SET<String>>();
      for (String filename : args) {
         In in = new In(filename);
         while (!in.isEmpty()) {
            String word = in.readString();
            if (!st.contains(word))
               st.put(word, new SET<String>());
            st.get(word).add(filename);
         }
      }
      while (!Stdin.isEmpty()) {
         String query = StdIn.readString();
         System.out.println(st.get(query));
      }
   }
}
```

build inverted index

process queries

Inverted Index: Example

Ex. Index all your `.java` files.

```
% java InvertedIndex *.java
set
DeDup.java ExceptionFilter.java InvertedIndex.java SET.java

vector
SparseVector.java SparseMatrix.java

spotlight
NOT FOUND
```


Inverted Index

Extensions.

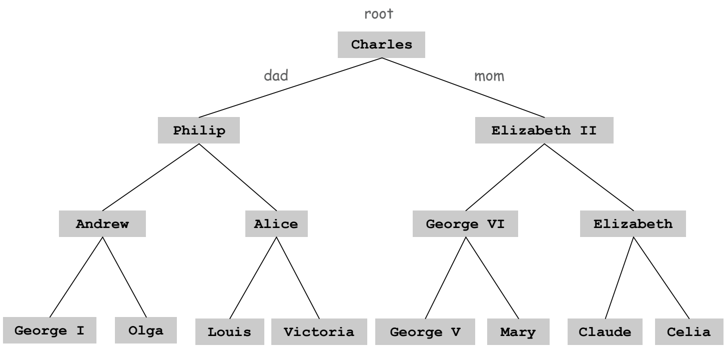
- Ignore case.
- Ignore stopwords: `the`, `on`, `of`, …
- Boolean queries: set intersection (AND), set union (OR).
- Proximity search: multiple words must appear nearby.
- Record position and number of occurrences of word in document.

Other Types of Trees

Other Types of Trees

Other types of trees.

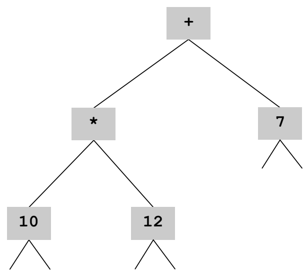
- Ancestor tree.



Other Types of Trees

Other types of trees.

- Ancestor tree.
- Parse tree: represents the syntactic structure of a statement, sentence, or expression.

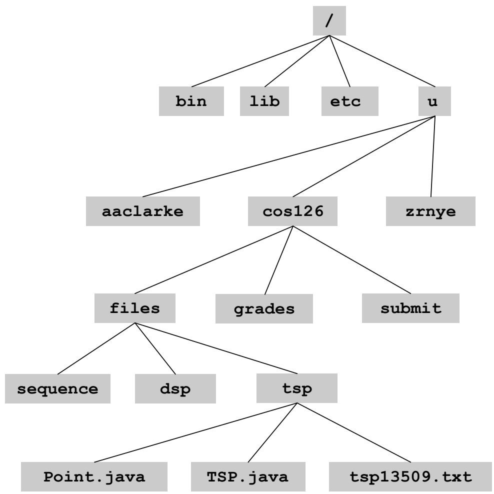


(10 * 12) + 7

Other Types of Trees

Other types of trees.

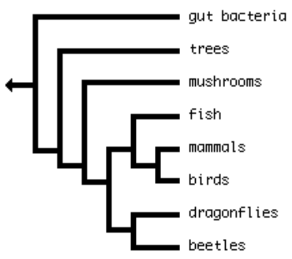
- Ancestor tree.
- Parse tree.
- Unix file hierarchy.



Other Types of Trees

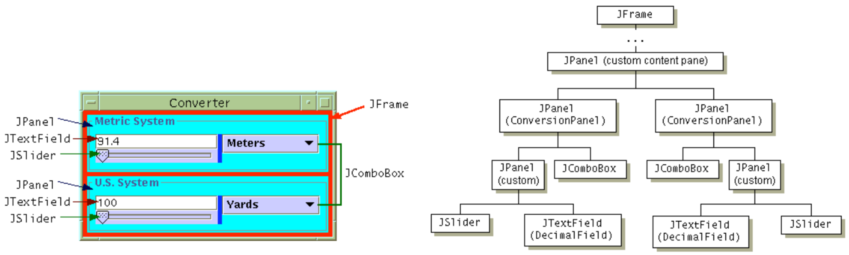
Other types of trees.

- Ancestor tree.
- Parse tree.
- Unix file hierarchy.
- Phylogeny tree.



Other types of trees.

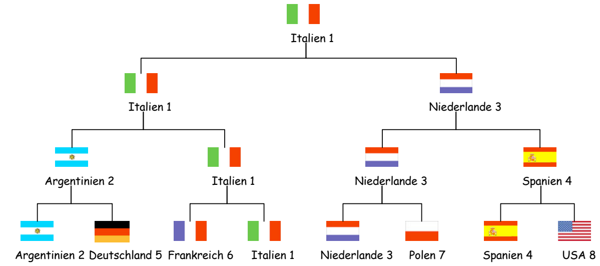
- Ancestor tree.
- Parse tree.
- Unix file hierarchy.
- Phylogeny tree.
- GUI containment hierarchy.



Reference: http://java.sun.com/docs/books/tutorial/uiswing/overview/anatomy.html

Other types of trees.

- Ancestor tree.
- Parse tree.
- Unix file hierarchy.
- Phylogeny tree.
- GUI containment hierarchy.
- Tournament trees.



Reference: Tobias Lauer