```
// Project 3: AnagramTree
 1
 2
    // BST's are confusing.
 3
    // Andrea Smith
 4
    // CSCI 1913
 5
 6
    import java.io.FileReader; // Read Unicode chars from a file.
    import java.io.IOException; // In case there's IO trouble.
7
 8
 9
    class AnagramTree
10
    {
      private class TreeNode
11
12
13
        private byte[] summary; // the key
        private WordNode words; // the value
14
15
        private TreeNode left;
        private TreeNode right;
16
17
        private TreeNode(String words, byte[] summary)
18
19
        {
20
          this.summary = summary;
21
          this.words = new WordNode(words, null);
          left = null;
22
          right = null;
23
24
        }
25
      }
26
27
      TreeNode head;
28
29
      private class WordNode
30
31
        private String word;
32
        private WordNode next;
33
34
        private WordNode(String word, WordNode next)
35
        {
          this.word = word;
36
37
          this.next = next;
38
        }
39
      }
40
41
      public AnagramTree()
42
      {
43
        head = new TreeNode(null, null);
44
      }
45
      public void add(String word)
46
      {
47
```

```
// build the anagram tree
48
        TreeNode foo = head;
49
50
        TreeNode bar = foo.right;
51
        byte [] temp = new byte [26];
        temp = stringToSummary(word);
52
        boolean needAdd = false;
53
        boolean left = false;
54
55
        boolean exists = false;
56
        while (bar != null)
57
58
        {
          int summ = compareSummaries(bar.summary, temp);
59
60
          if (summ < 0) // goes right
61
          {
            foo = bar;
62
            bar = bar.right;
63
             left = false;
64
65
          }
66
          else if (summ > 0) // goes left
67
          {
68
             foo = bar;
            bar = bar.left;
69
70
            left = true;
          }
71
          else // checks for duplicates
72
73
74
            WordNode exNode = bar.words;
75
            while (exNode != null)
76
             {
77
               if (exNode.word.equals(word))
78
               {
79
                 exists = true;
80
                 break;
               }
81
               exNode = exNode.next;
82
            }
83
84
85
             if (!exists) // if word wasn't there already, stick 'em in.
             {
86
               bar.words = new WordNode(word, bar.words);
87
88
             }
89
             needAdd = true;
             break;
90
          }
91
        }
92
93
\cap A
        if (15554744)
```

```
94
         II (:neeuAuu)
 95
         {
 96
           if (!left)
97
           {
98
             foo.right = new TreeNode(word, temp);
           }
99
100
           else
101
           {
             foo.left = new TreeNode(word, temp);
102
103
           }
         }
104
105
106
       }
107
       public void anagrams()
108
109
110
         orderGram(head.right);
111
       }
112
       private void orderGram(TreeNode thisNode)
113
114
       {
         if(thisNode != null)
115
116
           orderGram(thisNode.left);
117
           orderGram(thisNode.right);
118
           if (thisNode.words.next != null)
119
120
             System.out.println(); // So anagrams don't print as one line
121
             while (thisNode.words != null)
122
123
                System.out.print(thisNode.words.word + " ");
124
                thisNode.words = thisNode.words.next;
125
126
             }
           }
127
128
         }
129
       }
130
131
       private int compareSummaries(byte [] left, byte[] right)
132
         for (int i = 0; i < 26; i++)
133
134
         {
135
           if (left[i] != right[i])
136
              return left[i] - right[i];
137
           }
138
139
         }
         // will only get here if left already equals right
140
```

```
141
        return 0;
142
       }
143
144
       private byte[] stringToSummary(String word)
145
         byte[]foo = new byte[26];
146
147
         for (int i = 0; i < word.length(); i++)</pre>
148
149
             foo[word.charAt(i) - 'a']++;
150
         }
        return foo;
151
152
       }
     }
153
154
155
     class Anagrammer
156
157
       public static void main(String [] args)
158
159
         AnagramTree grams = new AnagramTree();
         Words words = new Words(args[0]);
160
161
162
        while (words.hasNext())
163
164
           grams.add(words.next());
165
         }
166
167
         grams.anagrams();
      }
168
     }
169
170
171
     //
172
     //
        WORDS. An iterator that reads lower case words from a text file.
     //
173
         James Moen
174
     //
175
     //
          19 Apr 17
176
    //
177
     // WORDS. Iterator. Read words, represented as STRINGs, from a text
178
     file. Each
     // word is the longest possible contiguous series of alphabetic
179
     ASCII CHARS.
180
    class Words
181
182
183
    private int
                          ch; // Last CHAR from READER, as an
      INT.
184
       private FileReader reader; // Read CHARs from here.
```

```
185
      private StringBuilder word; // Last word read from READER.
186
     // Constructor. Initialize an instance of WORDS, so it reads words
187
     from a file
     // whose pathname is PATH. Throw an exception if we can't open PATH.
188
189
190
       public Words(String path)
191
       {
192
         try
193
         {
194
           reader = new FileReader(path);
           ch = reader.read();
195
         }
196
         catch (IOException ignore)
197
198
           throw new IllegalArgumentException("Cannot open '" + path +
199
           "'."):
200
         }
       }
201
202
203
     // HAS NEXT. Try to read a WORD from READER, converting it to lower
     case as we
•
     // go. Test if we were successful.
204
205
206
       public boolean hasNext()
207
         word = new StringBuilder();
208
         while (ch > 0 && ! isAlphabetic((char) ch))
209
210
           read();
211
212
         }
         while (ch > 0 && isAlphabetic((char) ch))
213
214
           word.append(toLower((char) ch));
215
216
           read();
217
         }
218
         return word.length() > 0;
219
       }
220
221
     // IS ALPHABETIC. Test if CH is an ASCII letter.
222
223
       private boolean isAlphabetic(char ch)
224
       {
         return 'a' <= ch && ch <= 'z' || 'A' <= ch && ch <= 'Z';
225
226
       }
227
228
     // NFXT If HAS NFXT is true then return a WORD read from RFADER
```

```
// NEATE IT THE MEAT IS CLUC, CHELL LOCALLE A MORE LOCALLED ALL MEATER
     as a STRING.
     // Otherwise, return an undefined STRING.
229
230
231
     public String next()
232
233
     return word.toString();
234
235
236
     // READ. Read the next CHAR from READER. Set CH to the CHAR,
     represented as an
237
     // INT. If there are no more CHARs to be read from READER, then set
     CH to -1.
238
     private void read()
239
240
241
         try
242
        ch = reader.read();
243
244
        }
245
        catch (IOException ignore)
246
        {
247
           ch = -1;
         }
248
249
     }
250
251
     // TO LOWER. Return the lower case ASCII letter which corresponds
     to the ASCII
    // letter CH.
252
253
     private char toLower(char ch)
254
255
256
         if ('a' \leq ch && ch \leq 'z')
257
         {
258
           return ch;
259
         }
260
        else
261
           return (char) (ch - 'A' + 'a');
262
263
         }
264
      }
265
266
     }
267
```