```java
// Project 3: AnagramTree
// BST's are confusing.
// Andrea Smith
// CSCI 1913

import java.io.FileReader;   //  Read Unicode chars from a file.
import java.io.IOException;  //  In case there's IO trouble.

class AnagramTree
{
  private class TreeNode
  {
    private byte[] summary; // the key
    private WordNode words; // the value
    private TreeNode left;
    private TreeNode right;

    private TreeNode(String words, byte[] summary)
    {
      this.summary = summary;
      this.words = new WordNode(words, null);
      left = null;
      right = null;
    }
  }

  TreeNode head;

  private class WordNode
  {
    private String word;
    private WordNode next;

    private WordNode(String word, WordNode next)
    {
      this.word = word;
      this.next = next;
    }
  }

  public AnagramTree()
  {
    head = new TreeNode(null, null);
  }

  public void add(String word)
  {
    // build the anagram tree
    TreeNode foo = head;
    TreeNode bar = foo.right;
    byte [] temp = new byte [26];
    temp = stringToSummary(word);
    boolean needAdd = false;
    boolean left = false;
    boolean exists = false;

    while (bar != null)
    {
      int summ = compareSummaries(bar.summary, temp);
      if (summ < 0) // goes right
      {
        foo = bar;
        bar = bar.right;
        left = false;
      }
      else if (summ > 0) // goes left
      {
        foo = bar;
        bar = bar.left;
        left = true;
      }
      else // checks for duplicates
      {
        WordNode exNode = bar.words;
        while (exNode != null)
        {
          if (exNode.word.equals(word))
          {
            exists = true;
            break;
          }
          exNode = exNode.next;
        }

        if (!exists) // if word wasn't there already, stick 'em in.
        {
          bar.words = new WordNode(word, bar.words);
        }
        needAdd = true;
        break;
      }
    }

    if (!needAdd)
    if (!needAdd)
    {
      if (!left)
      {
        foo.right = new TreeNode(word, temp);
      }
      else
      {
        foo.left = new TreeNode(word, temp);
      }
    }

  }

  public void anagrams()
  {
    orderGram(head.right);
  }

  private void orderGram(TreeNode thisNode)
  {
    if(thisNode != null)
    {
      orderGram(thisNode.left);
      orderGram(thisNode.right);
      if (thisNode.words.next != null)
      {
        System.out.println(); // So anagrams don't print as one line
        while (thisNode.words != null)
        {
          System.out.print(thisNode.words.word + " ");
          thisNode.words = thisNode.words.next;
        }
      }
    }
  }

  private int compareSummaries(byte [] left, byte[] right)
  {
    for (int i = 0; i < 26; i++)
    {
      if (left[i] != right[i])
      {
        return left[i] - right[i];
      }
    }
    // will only get here if left already equals right
```

```java
    return 0;
  }

  private byte[] stringToSummary(String word)
  {
    byte[]foo = new byte[26];
    for (int i = 0; i < word.length(); i++)
    {
        foo[word.charAt(i) - 'a']++;
    }
    return foo;
  }
}

class Anagrammer
{
  public static void main(String [] args)
  {
    AnagramTree grams = new AnagramTree();
    Words words = new Words(args[0]);

    while (words.hasNext())
    {
      grams.add(words.next());
    }

    grams.anagrams();
  }
}

//
//  WORDS. An iterator that reads lower case words from a text file.
//
//    James Moen
//    19 Apr 17
//

//  WORDS. Iterator. Read words, represented as STRINGs, from a text file. Each
//  word is the longest possible contiguous series of alphabetic ASCII CHARs.

class Words
{
  private int           ch;      //  Last CHAR from READER, as an INT.
  private FileReader    reader;  //  Read CHARs from here.
  private StringBuilder word;    //  Last word read from READER.

//  Constructor. Initialize an instance of WORDS, so it reads words from a file
//  whose pathname is PATH. Throw an exception if we can't open PATH.

  public Words(String path)
  {
    try
    {
      reader = new FileReader(path);
      ch = reader.read();
    }
    catch (IOException ignore)
    {
      throw new IllegalArgumentException("Cannot open '" + path + "'.");
    }
  }

//  HAS NEXT. Try to read a WORD from READER, converting it to lower case as we
//  go. Test if we were successful.

  public boolean hasNext()
  {
    word = new StringBuilder();
    while (ch > 0 && ! isAlphabetic((char) ch))
    {
      read();
    }
    while (ch > 0 && isAlphabetic((char) ch))
    {
      word.append(toLower((char) ch));
      read();
    }
    return word.length() > 0;
  }

//  IS ALPHABETIC. Test if CH is an ASCII letter.

  private boolean isAlphabetic(char ch)
  {
    return 'a' <= ch && ch <= 'z' || 'A' <= ch && ch <= 'Z';
  }

//  NEXT. If HAS NEXT is true, then return a WORD read from READER
//  NEXT. If HAS NEXT is true, then return a WORD read from READER as a STRING.
//  Otherwise, return an undefined STRING.

  public String next()
  {
    return word.toString();
  }

//  READ. Read the next CHAR from READER. Set CH to the CHAR, represented as an
//  INT. If there are no more CHARs to be read from READER, then set CH to -1.

  private void read()
  {
    try
    {
      ch = reader.read();
    }
    catch (IOException ignore)
    {
      ch = -1;
    }
  }

//  TO LOWER. Return the lower case ASCII letter which corresponds to the ASCII
//  letter CH.

  private char toLower(char ch)
  {
    if ('a' <= ch && ch <= 'z')
    {
      return ch;
    }
    else
    {
      return (char) (ch - 'A' + 'a');
    }
  }

}
```