Homework Turnin

Name:	Ameya Singh
Account:	ameyas (ameyas@uw.edu)
Student ID:	1868457
Section:	AQ
Course:	CSE 143 18au
Assignment:	a8
Receipt ID:	7b4c8330b79e46bfdc87b54fd5cfeca5

Turnin Successful!

The following file(s) were received:

HuffmanNode.java (1920 bytes)

```
 1. /*
 2.  * Author: Ameya Singh
 3.  * CSE 143 AQ
 4.  * TA: Soham P.
 5.  * Homework 8: Huffman Code
 6.  */
 7.
 8. /**
 9.  * HuffmanNode provides a node class used in creating instances of HuffmanTree.
10.  */
11. public class HuffmanNode implements Comparable<HuffmanNode> {
12.     /**
13.      * Nullable object holding integer representation of character.
14.      */
15.     public Integer character;
16.     /**
17.      * Nullable object holding integer number of occurrences of character.
18.      */
19.     public Integer numOccurrences;
20.     /**
21.      * Reference to left child node.
22.      */
23.     public HuffmanNode left;
24.     /**
25.      * Reference to right child node.
26.      */
27.     public HuffmanNode right;
28.
29.     /**
30.      * Constructs new HuffmanNode with no children.
31.      * @param character Character node represents.
32.      * @param numOccurrences Occurrences of represented character.
33.      */
34.     public HuffmanNode(Integer character, Integer numOccurrences) {
35.         this(character, numOccurrences, null, null);
36.     }
37.
38.     /**
39.      * Constructs a new HuffmanNode.
40.      * @param character Character node represents.
41.      * @param numOccurrences Occurrences of represented character.
42.      * @param left Left child node.
43.      * @param right Right child node.
44.      */
45.     public HuffmanNode(Integer character, Integer numOccurrences,
46.                        HuffmanNode left, HuffmanNode right) {
47.         this.character = character;
48.         this.numOccurrences = numOccurrences;
49.         this.left = left;
```

```
50.         this.right = right;
51.     }
52.
53.     /**
54.      * Compares number of occurrences.
55.      * @param o Other HuffmanNode.
56.      * @return Whether node has greater than less than or equal numOccurrences.
57.      */
58.     @Override
59.     public int compareTo(HuffmanNode o) {
60.         if (this.numOccurrences > o.numOccurrences) {
61.             return 1;
62.         } else if (this.numOccurrences < o.numOccurrences) {
63.             return -1;
64.         } else {
65.             return 0;
66.         }
67.     }
68. }
69.
70.
```

HuffmanTree.java (4721 bytes)

```
1. /*
2.  * Author: Ameya Singh
3.  * CSE 143 AQ
4.  * TA: Soham P.
5.  * Homework 8: Huffman Code
6.  */
7.
8. import java.io.*;
9. import java.util.*;
10.
11. /**
12.  * HuffmanTree provides a framework for compressing text files using the Huffman
13.  * coding scheme.
14.  */
15. public class HuffmanTree {
16.     /**
17.      * Holds a reference to the root node of the HuffmanTree.
18.      */
19.     private HuffmanNode root;
20.
21.     /**
22.      * Constructs a new HuffmanTree using the passed array of the frequency of
23.      * characters.
24.      * @param count Integer array where count[i] is the number of occurrences of
25.      *              the character with integer value i.
26.      */
27.     public HuffmanTree(int[] count) {
28.         Queue<HuffmanNode> priorityQueue = new PriorityQueue<HuffmanNode>();
29.
30.         for (int i = 0; i < count.length; i++) {
31.             if (count[i] > 0) {
32.                 priorityQueue.add(new HuffmanNode(i, count[i]));
33.             }
34.         }
35.         priorityQueue.add(new HuffmanNode(count.length, 1));
36.
37.         while (priorityQueue.size() > 1) {
38.             HuffmanNode node1 = priorityQueue.remove();
39.             HuffmanNode node2 = priorityQueue.remove();
40.             HuffmanNode newNode = new HuffmanNode(null,
41.                     node1.numOccurrences + node2.numOccurrences, node1, node2);
42.             priorityQueue.add(newNode);
43.         }
44.
45.         root = priorityQueue.remove();
46.     }
47.
48.     /**
49.      * Reconstructs a HuffmanTree from file.
50.      * @param input Scanner containing tree in standard format.
51.      */
52.     public HuffmanTree(Scanner input) {
53.         while (input.hasNextLine()) {
54.             int character = Integer.parseInt(input.nextLine());
```

```
55.             String pathToNode = input.nextLine();
56.             root = read(root, character, pathToNode);
57.         }
58.     }
59.
60.     /**
61.      * Private helper used to assist reading in HuffmanTrees.
62.      * @param node Current node.
63.      * @param character Current character to represent in node.
64.      * @param pathToNode Current path to node.
65.      * @return Returns node conforming to passed parameters.
66.      */
67.     private HuffmanNode read(HuffmanNode node, int character,
68.                              String pathToNode) {
69.         if (pathToNode.isEmpty()) {
70.             return new HuffmanNode(character, null);
71.         } else {
72.             if (node == null) {
73.                 node = new HuffmanNode(null, null);
74.             }
75.
76.             if (pathToNode.charAt(0) == '0') {
77.                 node.left = read(node.left, character, pathToNode.substring(1));
78.             } else {
79.                 node.right = read(node.right, character,
80.                         pathToNode.substring(1));
81.             }
82.         }
83.         return node;
84.     }
85.
86.     /**
87.      * Writes HuffmanTree in standard format to passed output stream.
88.      * @param output PrintStream to which to write tree.
89.      */
90.     public void write(PrintStream output) {
91.         write(output, root, "");
92.     }
93.
94.     /**
95.      * Private helper for writing tree to file.
96.      * @param output PrintStream to write to.
97.      * @param node Current node.
98.      * @param pathToNode Current path taken.
99.      */
100.    private void write(PrintStream output, HuffmanNode node,
101.                       String pathToNode) {
102.        if (node != null) {
103.            if (node.left == null && node.right == null) {
104.                output.println(node.character);
105.                output.println(pathToNode);
106.            }
107.            write(output, node.left, pathToNode + "0");
108.            write(output, node.right, pathToNode + "1");
109.        }
110.    }
111.
112.    /**
113.     * Reads bits from passed input stream and writes corresponding characters
114.     * from the HuffmanTree to the given output stream. Will stop reading when
115.     * a character matching the passed end of file parameter is reached.
116.     * @param input BitInputStream containing encoded characters to decode.
117.     * @param output PrintStream to which decoded characters will be written.
118.     * @param eof Character representing position at which to stop reading file.
119.     */
120.    public void decode(BitInputStream input, PrintStream output, int eof) {
121.        int currentBit = input.readBit();
122.        HuffmanNode currNode = root;
123.        boolean reachedEOF = false;
124.
125.        while (!reachedEOF) {
126.            if (currNode.left == null && currNode.right == null) {
127.                if (currNode.character == eof) {
128.                    reachedEOF = true;
129.                } else {
130.                    output.write(currNode.character);
131.                    currNode = root;
132.                }
133.            }
134.
135.            if (currentBit == 0) {
```

```
136.                 currNode = currNode.left;
137.             } else {
138.                 currNode = currNode.right;
139.             }
140.
141.             currentBit = input.readBit();
142.         }
143.     }
144. }
145.
```