

```

1 import java.io.IOException;
2
3
4
5
6
7 public class HuffmanTree {
8
9     private HNode root;
10
11     /**
12      * build a Huffman tree using the given characters and corresponding frequencies
13      * @param frequencies the corresponding frequencies of given character
14      */
15     public HuffmanTree(TreeMap<Character, Integer> frequencies){
16         PriorityQueue<HNode> pq = new PriorityQueue<HNode>(frequencies.size(), new
HNodeComparator<HNode>());
17         for (Entry<Character, Integer> entry : frequencies.entrySet()){
18             HNode leaf = new HNode(entry.getKey(), entry.getValue());
19             pq.add(leaf);
20         }
21         int pqSize = pq.size();
22         for (int i = 1; i < pqSize; i++){
23             HNode h1 = (HNode) pq.poll();
24             HNode h2 = (HNode) pq.poll();
25             HNode h3 = new HNode(h1, h2);
26             pq.add(h3);
27         }
28         root = (HNode) pq.poll();
29     }
30
31     /**
32      * binary encoding of the given symbol using binary characters '0' and '1'
33      * @param symbol the given symbol
34      * @return the string of binary encoding
35      */
36     public String encodeLoop(char symbol){
37         String encoded = "";
38         HNode curr = root;
39         while (!curr.isLeaf()){
40             if (curr.leftChild.contains(symbol)){
41                 encoded = encoded + "0";
42                 curr = curr.leftChild;
43             }
44             else if (curr.rightChild.contains(symbol)){
45                 encoded = encoded + "1";
46                 curr = curr.rightChild;
47             }
48         }
49         return encoded;
50     }
51
52     /**
53      * recursive method
54      * binary encoding of the given symbol using binary characters '0' and '1'
55      * @param symbol the given symbol
56      * @return the string of binary encoding
57      */
58     public String encode(char symbol){
59         return encode(symbol, root);
60     }
61 }
62

```

HuffmanTree.java

```

63
64 /**
65  * recursive method
66  * binary encoding of the given symbol using binary characters '0' and '1' from HNode node
67  * @param symbol the given symbol
68  * @param node the node to start encoding
69  * @return the string of binary encoding
70  */
71 private String encode(char symbol, HNode node){
72     if (node.isLeaf()){
73         return "";
74     }
75     else if (node.leftChild.contains(symbol)){
76         return "0" + encode(symbol, node.leftChild);
77     }
78     else if (node.rightChild.contains(symbol)){
79         return "1" + encode(symbol, node.rightChild);
80     }
81     else{
82         throw new NoSuchElementException();
83     }
84 }
85
86 /**
87  * decode the code
88  * @param code the given code
89  * @return the symbol of corresponding to the given code
90  */
91 public char decode(String code){
92     HNode curr = root;
93     for (char i : code.toCharArray()){
94         if (i == '0'){
95             if (curr.leftChild != null){
96                 curr = curr.leftChild;
97             }
98             else{
99                 return '\0';
100             }
101         }
102         if (i == '1'){
103             if (curr.rightChild != null){
104                 curr = curr.rightChild;
105             }
106             else{
107                 return '\0';
108             }
109         }
110     }
111     if (curr.symbolSet.length() == 1){
112         return curr.symbolSet.charAt(0);
113     }
114     return '\0';
115 }
116
117 /**
118  * write the individual bits of the given symbol using encoding to bitOutputStream
119  * @param symbol the given symbol
120  * @param stream the output stream
121  * @return return true if written success

```

HuffmanTree.java

```

122     */
123     public boolean writeCode(char symbol, BitOutputStream stream){
124         HNode curr = root;
125         while (!curr.isLeaf()){
126             if (curr.leftChild.contains(symbol)){
127                 try {
128                     stream.writeBit(1);
129                 } catch (IOException e) {
130                     e.printStackTrace();
131                 }
132                 curr = curr.leftChild;
133             }
134             else if (curr.rightChild.contains(symbol)){
135                 try {
136                     stream.writeBit(0);
137                 } catch (IOException e) {
138                     e.printStackTrace();
139                 }
140                 curr = curr.rightChild;
141             }
142         }
143         return true;
144     }
145
146
147     /**
148     * read the next symbol of binary encoding individual bits from BitInputStream
149     * and return the corresponding characters
150     * @param stream the input stream
151     * @return the corresponding character
152     */
153     public char readCode(BitInputStream stream){
154         HNode curr = root;
155         int b = -1;
156         try {
157             b = stream.readBit();
158         } catch (IOException e) {
159             e.printStackTrace();
160         }
161         while(b != -1){
162             if (b == '0'){
163                 if (curr.leftChild != null){
164                     curr = curr.leftChild;
165                 }
166             }
167             else{
168                 return '\0';
169             }
170             if (b == '1'){
171                 if (curr.rightChild != null){
172                     curr = curr.rightChild;
173                 }
174             }
175             else{
176                 return '\0';
177             }
178         }
179         if (curr.symbolSet.length() == 1){
180             return curr.symbolSet.charAt(0);

```

HuffmanTree.java

```
181     }  
182     return '\0';  
183 }  
184  
185 public HNode getRoot(){  
186     return root;  
187 }  
188  
189 }  
190
```