Nov 17, 20 16:22 **HEncode.java** Page 1/9

```
/*Alan Stoloff
    * Data Structures
2
    * Dr. Benjamin
4
          class HEncode - A program to encode a file using Huffman Code Algorithm.
5
    * /
6
   import java.io.*;
8
   import java.lang.reflect.Array;
9
   public class HEncode {
10
11
       private Node root = null;
                                                   // Root of the Huffman Code Tree.
12
13
       private Node[] leafPtr = new Node[256]; // Array of pointers to leaf nodes.
14
                                                   // used to traverse up the code
15
16
                                                    // tree during the encoding process.
17
       private int[] freq = new int[256]; // Frequency of the bytes being encoded.
18
                                                    used to build the initial trees
                                              //
19
                                              //
                                                    when building the code tree.
20
21
       private String inputFilename;
                                           // The name of the file to encode.
22
23
                           // pq is a priority queue of root nodes to trees.
24
                           // is used during the building of the code tree to
25
                           // select the roots with minimum frequency count.
26
27
       private PriorityQueue<Node> pq = new PriorityQueue<Node>();
28
29
                           // stk is a stack used to store the 0's and 1's
                           // of the code for a byte being encoded. As a byte
31
                           // is being encoded, we travel from a leaf node, up
32
33
                           // the parent pointers to the root, pushing 0's and 1's
                           // as appropriate for the encoding. When the root // is reached, we pop the stack of "bits" into the
34
35
36
                           // output file.
37
       private Stack<Integer> stk = new Stack<Integer>();
38
39
       private BitWriter bitw; // Writes bits to the outputfile.
40
41
       public final boolean DEBUG = true; // When true debugging info
42
                                              // is displayed.
43
44
45
46
```

Page 2/9

Nov 17, 20 16:22 **HEncode.java**

```
46
47
48
       public static void main(String[] args)
49
                throws FileNotFoundException, IOException
50
            if (args.length != 1)
51
                System.out.println("Incorrect program argument");
52
                System.exit(0);
53
54
55
           HEncode coder = new HEncode(args[0]); // Construct a Huffman Encoder
56
57
            coder.getFrequencies(); // Get the frequencies of bytes in inputfile.
58
            if (coder.DEBUG)
59
                                      // For debug - Let's see if we got the freqs.
                coder.showFreq();
60
61
           coder.getLeafPtrs();
                                      // Get initial trees used to build code tree.
            if (coder.DEBUG)
                                      // For debug - print priority queue of roots.
62
                coder.showPQ();
63
           coder.buildTree();
                                      // Build the code tree.
64
                                      // For debug - print the code tree.
            if (coder.DEBUG)
65
                coder.printTree();
66
67
           coder.encodeFile();
                                      // Read the inputfile a second time, encoding
                                      // the inputfile.
68
       }
69
70
71
              Constructor - The argument is the name of the file to encode.
72
73
74
       public HEncode(String inputFilename)
75
76
            this.inputFilename = inputFilename;
77
78
79
80
```

Nov 17, 20 16:22 **HEncode.java** Page 3/9

```
80
81
             getFrequencies() - Open the given file and determine the frequency
82
                                  with which each byte (character) occurs.
83
84
             The frequencies are store in the array freq at the index
85
86
             location corresponding to the byte value 0 to 255.
         * /
87
88
        public void getFrequencies()
89
90
                                     // File object to read from.
            FileInputStream inF;
92
            int nextByte;
                                      // Next byte from the file.
93
            // Initialize the frequencies
94
95
            for (int i = 0; i < 256; i++)
96
                 freq[i] = 0;
97
98
99
                 inF = new FileInputStream(inputFilename); // Open the input file.
100
101
102
                 do {
                     nextByte = inF.read();
                                                  // Read the next byte (-1 on EOF)
103
104
                     if (nextByte != -1)
                                                 //
                         freq[nextByte]++;
                                                  //
                                                        Increment frequency counter
105
                 \} while (nextByte != -1);
                                                  //
106
                                                        for the byte.
107
                 inF.close();
                                                  // Close the file.
108
109
            catch (FileNotFoundException e) {
                System.out.printf("Error opening file %s\n", inputFilename);
111
                System.exit(0);
112
113
            catch (IOException e) {
114
                 System.out.printf("IOException reading from: %s\n", inputFilename);
115
116
                 System.exit(0);
117
        }
118
119
120
121
122
             showFreq() - display the byte frequency array.
123
124
125
             For debugging purposes we want to show the frequency with
             which each byte (or character) occurs.
126
127
128
        public void showFreq()
129
130
            for (int i = 0; i < 256; i++) {
                                                 // Only show the bytes
131
                 if (freq[i] != 0)
                                                  // having non-zero frequency.
132
                     System.out.printf("byte: %3d char: %c freq: %d\n",
133
                                                       i, (char) i, freq[i]);
134
135
136
137
```

HEncode.java Nov 17, 20 16:22 Page 4/9 137 138 getLeafPtrs() - Create root nodes containing the bytes. These nodes 139 140 are the roots of the initial trees used to build the code tree. They will become the leaf nodes of the 141 Huffman Code tree. 142 143 Enter the nodes in the priority queue of roots. 144 as each node is created. 145 * Nodes with smaller frequency have higher priority. 146 * / 147 148 149 public void getLeafPtrs() 150 //need a ponter to priority Queue 151 for(int i=0;i<256;i++){ 152 if(freq[i]!=0){ 153 Node temp=new Node(); 154 leafPtr[i]=temp; 155 temp.data=(byte)i; 156 temp.frequency=freq[i]; 157 pq.enqueue(temp); 158 159 160 161 162 public void showPQ() 163 164 System.out.println(pq.toString()); 165 166 167 168 buildTree() - A function to build the Huffman Code Tree. 169 We start with a priority queue of the leaf nodes. 170 171 172 173 public void buildTree() 174 Boolean keepGoing=true; 175 176 Node temp=pq.dequeue(); while(keepGoing){ 178 if(pq.isEmpty()){ pq.enqueue(temp); 179 this.root=temp; 180 keepGoing=false; 181 break; 182 183 Node rootTemp=new Node(); 184 Node rtemp=pq.dequeue(); 185 rootTemp.rchild=rtemp; 186 187 rootTemp.lchild=temp; temp.parent=rootTemp; 188 rtemp.parent=rootTemp; 189 rootTemp.frequency=rootTemp.rchild.frequency+rootTemp.lchild.frequency; 190 temp=rootTemp; 191 192 193 194 195 encodeFile() - a function to encode a file - that is to 196 create a compressed file using the Huffman 197 algorithm. The function: 198 1. Creates a BitWriter to write the compressed file. 199 200 2. Uses the BitWriter to write the size of the original file. 201 Calls writeTree() to write the code tree. 4. Opens the original file. 202 5. Repeatedly reads a byte from the file and 203 and calls writeCode to write the coded bits 204 for the byte to the compressed file.

205

Nov 17, 20 16:22 **HEncode.java** Page 5/9

```
6. Closes the BitWriter
206
207
208
209
       public void encodeFile()
210
           bitw=new BitWriter("book.txt.huf");
211
212
           bitw.writeInt(root.frequency);
213
           writeTree(root);
           FileInputStream inF;
214
           int nextByte;
215
216
           try
               inF = new FileInputStream(inputFilename); // Open the input file.
217
218
               do {
219
                   nextByte = inF.read();
                                             // Read the next byte (-1 on EOF)
220
221
                   if (nextByte != -1)
                                             //
                       writeCode((byte)nextByte);
                                                      //
                                                            Increment frequency counter
222
               } while (nextByte != -1);
                                           //
                                                   for the byte.
223
224
               inF.close();
225
                                               // Close the file.
               bitw.close();
226
227
           catch (FileNotFoundException e) {
228
               229
230
               System.exit(0);
231
           catch (IOException e) {
232
               System.out.printf("IOException reading from: %s\n", inputFilename);
233
               System.exit(0);
234
235
237
```

Nov 17, 20 16:22 **HEncode.java** Page 6/9

```
237
238
239
240
             writeCode() - A function to encode byte b. The function
                             uses b as an index into the array of pointers
241
                 to leaf nodes of the Huffman Code Tree. Once at the leaf,
242
243
                 parent pointers are used to climb to the root, pushing
                 0's and 1's on a stack according to the encoding. Once
244
                 at the root, the o's and 1's on the stack are popped off
245
                 and written to the outputfile using the bitWriter.
246
         * /
247
248
        public void writeCode(byte b)
249
250
            Node ptr=leafPtr[b];
251
252
            while(ptr!=root){
                 Node ptrParent=ptr.parent;
253
                 if(ptrParent.rchild==ptr){
254
                     stk.push(1);
255
256
                 if(ptrParent.lchild==ptr){
257
258
                     stk.push(0);
259
                ptr=ptr.parent;
260
261
            while(!stk.isEmpty()){
262
263
                bitw.writeBit(stk.pop());
264
265
266
267
              writeTree() - A recursive function to write the Huffman
                              Code Tree to the output file. The tree
268
                  must be stored with the encoded file so that it can
269
                  be used to decode the file.
270
271
272
273
        public void writeTree(Node root)
274
275
276
            if(root==null){
                 return;
277
278
            else if(root.lchild==null && root.rchild==null){
279
                 bitw.writeBit(0);
280
                bitw.writeByte(root.data);
281
282
            else{
283
                bitw.writeBit(1);
284
285
                writeTree(root.lchild);
                 writeTree(root.rchild);
286
287
288
289
290
```

Nov 17, 20 16:22 **HEncode.java** Page 7/9

```
290
291
             printTree() - Print the Huffman Code Tree to
293
                             standard output.
294
295
296
        public void printTree()
297
298
            rPrintTree(root,0);
299
300
301
302
              rPrintTree() - the usual quick recursive method to print a tree.
303
304
305
        public void rPrintTree(Node r, int level)
306
307
308
            if (r == null)
                                       // Empty tree.
309
310
311
            rPrintTree(r.rchild, level + 1); // Print the right subtree.
312
            for (int i = 0; i < level; i++)
313
314
                 System.out.print("
315
316
            if (r.data > (byte) 31)
317
                 System.out.printf("%c-%d\n", (char) r.data, r.frequency);
318
319
                 System.out.printf("%c-%d\n", '*', r.frequency);
321
            rPrintTree(r.lchild, level + 1);
322
323
324
325
```

```
HEncode.java
Nov 17, 20 16:22
                                                                                            Page 8/9
325
326
               Node - an inner class to represent a node of
327
328
                       a Huffman Code Tree.
         * /
329
330
331
        private class Node implements Comparable < Node >
332
             byte data;
                                    // A byte of data - stored in an Integer.
333
             Node lchild;
                                    // Left child pointer.
334
             Node rchild;
                                    // Right child pointer.
335
             Node parent;
                                    // Pointer to parent node.
336
337
             Integer frequency;
                                    // Frequency the data within
                                    // a file being encoded.
338
339
                  Basic node constructor.
340
341
342
             public Node()
343
344
                 data = 0;
                                       // Each Huffman Code Tree node
345
                 lchild = null;
                                      // contains data, pointers to
346
                 rchild = null;
                                      // children and parent nodes
347
                 parent = null;
                                      // plus a frequency count
348
349
                 frequency = 0;
                                       // associated with the data.
             }
350
351
352
                  Constructor specifying all values
353
                  of the node instance variables.
354
355
356
            public Node(byte data, Node lchild, Node rchild,
357
                                       Node parent, int frequency)
358
359
                 this.data = data;
360
                 this.lchild = lchild;
361
                 this.rchild = rchild;
362
                 this.parent = parent;
363
364
                 this.frequency = frequency;
             }
365
366
             /*
367
                   compareTo() - Compare two frequency values. We want Nodes
368
                                   with lower frequencies to have higher priority
369
370
                                   in the priority queue.
371
              * /
372
373
             public int compareTo(Node other)
374
375
                 if(this.frequency>other.frequency){
376
                      return -1;
377
378
379
                 else if(this.frequency<other.frequency){</pre>
                      return 1;
380
381
382
                 else{
                      return 0;
383
384
             }
385
386
             public String toString()
387
388
                 char ch = (char) this.data;
389
390
                 String str = "byte: " + data + " char: ";
391
392
```

393

if (data > (byte) 31)

Printed by Alan Stoloff – 2023 HEncode.java Nov 17, 20 16:22 Page 9/9 str = str + (char) data + " freq: " + frequency; 394 395 str = str + " " + " freq: " + frequency; 397 return str; 398 } 399 400 401 }