Ryan A. Margraf

WES 237B

Assignment 1

7/24/2022

Encoding Scheme

The encoding scheme I implemented can be divided into a few basic steps:

1. Determine the frequency of each input character
2. Sort the characters from least to greatest frequency
3. Create the Huffman tree using a priority queue
4. Traverse the tree and create a code table
5. Encode the input using the code table

Determining the frequencies is easily accomplished by doing a for loop over the length of the input buffer. Nodes for the Huffman tree are then created, with each assigned a value (0-255), it's corresponding frequency, and pointers to its left and right children (null for now). These nodes are then sorted using radix sort. The result is a minheap that is then used as a priority queue to create the Huffman tree. Two nodes are removed, and a new intermediate node is created with them as its children, then it is put back into the minheap. This process continues until there are no nodes left in the heap, and then building the Huffman tree is complete.

The tree is then traversed using a recursive algorithm which keeps track of the left and right branches with a vector. At the end of this process, every leaf node will have a binary sequence assigned to it. All that is left then is to encode each input byte by accessing its corresponding code sequence, then turning that into bits. Since this process is done deterministically, it is sufficient to store the frequency array in the encoded file.

The output consists of 256 32-bit integers for the frequency array immediately followed by the encoded bits. As a result, the overhead is exactly 1024 bytes regardless of the size of the input file. 32 bits was used because the extra overhead was minimal (< 1 KB), but it allows for the compression of much larger files.

The last byte may have some extra bits not filled by the encoded text. In these cases, the algorithm inserts extra 0s until a complete byte can be written to the output buffer. These bits are safely ignored by the decoding algorithm because it stops once it outputs the number of characters in the original file (determined by adding together all entries in the frequency array).

For decoding, the first three steps are the same, but instead of creating a code table, the tree is traversed using the input bits, and when a leaf node is reached, the character stored at that node is output.

Changes to Starting Code

Besides huffman.cpp, huffman.h was modified to add the declarations of the helper functions I made in Huffman.cpp, and to include the Vector class. The Makefile was modified to remove the references to OpenCV. The files main.cpp and main.h were left unmodified.

How To Run

1. Open a terminal
2. Navigate to the "Assignment_1" folder (where this .pdf is located)
3. Type "make" and press enter
4. The program can be run using a command of the format

./CODEC <input file> <code file> <output file>

For example, for an input file named "input.txt", an intermediate code output file named "code.txt", and a decoded output file named "output.txt", the command would be

./CODEC input.txt code.txt output.txt

Sample Output

The terminal output produced by the given input file is below. The first array shows the frequency table of the input file, and the second shows the code table

WES237B Assignment 1
Frequency Array:
0: 0
1: 0
2: 0
3: 0
4: 0
5: 0
6: 0
7: 0
8: 0
9: 0
10: 1
11: 0
12: 0
13: 0
14: 0

15: 0
16: 0
17: 0
18: 0
19: 0
20: 0
21: 0
22: 0
23: 0
24: 0
25: 0
26: 0
27: 0
28: 0
29: 0
30: 0
31: 0
32: 4999
33: 0
34: 0
35: 0
36: 0
37: 0
38: 0
39: 0
40: 0
41: 0
42: 0
43: 0
44: 666
45: 0
46: 773
47: 0
48: 0
49: 0
50: 0
51: 0
52: 0
53: 0
54: 0
55: 0
56: 0

57: 0
58: 0
59: 12
60: 0
61: 0
62: 0
63: 0
64: 0
65: 49
66: 0
67: 76
68: 62
69: 33
70: 45
71: 0
72: 0
73: 46
74: 0
75: 0
76: 4
77: 49
78: 95
79: 0
80: 151
81: 22
82: 0
83: 73
84: 0
85: 21
86: 72
87: 0
88: 0
89: 0
90: 0
91: 0
92: 0
93: 0
94: 0
95: 0
96: 0
97: 2144
98: 375

99: 1104
100: 714
101: 3266
102: 186
103: 398
104: 170
105: 2696
106: 30
107: 0
108: 1722
109: 1234
110: 1649
111: 1154
112: 666
113: 334
114: 1515
115: 2352
116: 2204
117: 2622
118: 384
119: 0
120: 0
121: 19
122: 0
123: 0
124: 0
125: 0
126: 0
127: 0
128: 0
129: 0
130: 0
131: 0
132: 0
133: 0
134: 0
135: 0
136: 0
137: 0
138: 0
139: 0
140: 0

141: 0
142: 0
143: 0
144: 0
145: 0
146: 0
147: 0
148: 0
149: 0
150: 0
151: 0
152: 0
153: 0
154: 0
155: 0
156: 0
157: 0
158: 0
159: 0
160: 0
161: 0
162: 0
163: 0
164: 0
165: 0
166: 0
167: 0
168: 0
169: 0
170: 0
171: 0
172: 0
173: 0
174: 0
175: 0
176: 0
177: 0
178: 0
179: 0
180: 0
181: 0
182: 0

183: 0
184: 0
185: 0
186: 0
187: 0
188: 0
189: 0
190: 0
191: 0
192: 0
193: 0
194: 0
195: 0
196: 0
197: 0
198: 0
199: 0
200: 0
201: 0
202: 0
203: 0
204: 0
205: 0
206: 0
207: 0
208: 0
209: 0
210: 0
211: 0
212: 0
213: 0
214: 0
215: 0
216: 0
217: 0
218: 0
219: 0
220: 0
221: 0
222: 0
223: 0
224: 0

225: 0
226: 0
227: 0
228: 0
229: 0
230: 0
231: 0
232: 0
233: 0
234: 0
235: 0
236: 0
237: 0
238: 0
239: 0
240: 0
241: 0
242: 0
243: 0
244: 0
245: 0
246: 0
247: 0
248: 0
249: 0
250: 0
251: 0
252: 0
253: 0
254: 0
255: 0
Code Table:
0:
1:
2:
3:
4:
5:
6:
7:
8:
9:

10: 1100010010000
11:
12:
13:
14:
15:
16:
17:
18:
19:
20:
21:
22:
23:
24:
25:
26:
27:
28:
29:
30:
31:
32: 101
33:
34:
35:
36:
37:
38:
39:
40:
41:
42:
43:
44: 111100
45:
46: 00010
47:
48:
49:
50:
51:

52:
53:
54:
55:
56:
57:
58:
59: 110001001001
60:
61:
62:
63:
64:
65: 110000001
66:
67: 110001000
68: 110000010
69: 1100000111
70: 1111110000
71:
72:
73: 1111110001
74:
75:
76: 1100010010001
77: 110000000
78: 111111001
79:
80: 11000011
81: 11000100111
82:
83: 110000101
84:
85: 11000100110
86: 110000100
87:
88:
89:
90:
91:
92:
93:

94:
95:
96:
97: 0110
98: 1111111
99: 10000
100: 111110
101: 001
102: 11111101
103: 000111
104: 11000101
105: 1110
106: 1100000110
107:
108: 0101
109: 11001
110: 0100
111: 10001
112: 111101
113: 1100011
114: 0000
115: 1001
116: 0111
117: 1101
118: 000110
119:
120:
121: 11000100101
122:
123:
124:
125:
126:
127:
128:
129:
130:
131:
132:
133:
134:
135:

178:
179:
180:
181:
182:
183:
184:
185:
186:
187:
188:
189:
190:
191:
192:
193:
194:
195:
196:
197:
198:
199:
200:
201:
202:
203:
204:
205:
206:
207:
208:
209:
210:
211:
212:
213:
214:
215:
216:
217:
218:
219:

220:
221:
222:
223:
224:
225:
226:
227:
228:
229:
230:
231:
232:
233:
234:
235:
236:
237:
238:
239:
240:
241:
242:
243:
244:
245:
246:
247:
248:
249:
250:
251:
252:
253:
254:
255:
Encoding complete
Decoding complete
SUCCESS

The beginning of the coded file produced with the provided input file is shown below. There are a lot of blank spaces at the beginning, which correspond to the zeros in the frequency array. Afterwards is seemingly random data, which is to be expected in a binary file with the redundancy removed.