Huffman Coding Project Technical Design

Victor W. Frye

Davenport University

Huffman Coding Project Technical Design

Overview

This program will compress and decompress an input ASCII text file using the Huffman coding algorithm.

Scope

This program will compress or decompress an input ASCII text file using the Huffman coding algorithm. Execution of the program will be done from the command line and determine whether to compress or decompress based on the input arguments.

The input file for compression will be read as-is and not converted in any way. The input file is assumed to be an ASCII text file and is not translated in any way. The output file for compression will be plain binary text. Additionally, a key file will be generated to decompress the file to its original state.

The input file for decompression will be assumed to be a compressed file and will not be translated in any way. The input file is assumed to be a encrypted ASCII text file with only binary representation of the original data. A key file will be required to decompress the input file. If no key file is found, the input file will be read as corrupted.

Usage

To use the program, two additional command line arguments will be needed compared to the standard executable calling. The first will either be “compress” or “decompress”, designating what action to perform upon the file. The second argument is the file path for the input file.

Note that the output file and key file are defaulted within the program. The output file will output as “output.txt” and the key file will output as “key.txt”.

Processing Outline

1. If the number of command line arguments is not two
    a. Display usage error message
    b. Exit
2. If the action route argument is invalid (not compress or decompress)
    a. Display usage error message
    b. Exit
3. If the input file does not exist
    a. Display input error message
    b. Exit
4. Initialize an array of character frequency objects
    a. For each element, create a character frequency object with the character set to the corresponding ASCII value based upon the index
    b. Set the initial character frequency count to zero for each element in the array
5. If action route argument is to compress
    a. Open the input file
        i. Read a character from the input file
        ii. While not at the end of the input file
            1. Increment the character frequency count using the ASCII value of the character as the index
            2. Read the next character
    b. Close the input file

c. Sort the array of character frequency objects using the built-in array sort method

d. Initialize a linked list of nodes of type character frequency

    i. For each element in the array of character frequency objects

        1. If the frequency of the character frequency object is not zero (0)

            a. Initialize a node with the character frequency object inserted as the root of the tree

            b. Add the newly created node to the end of the linked list

e. Construct a binary tree of the character frequency objects

    i. While the count of linked list nodes is less than one (1)

        1. If only two nodes remain in the linked list and the first node is larger than the last

            a. Swap the first and last nodes

        2. Remove the first two (2) nodes from the linked list

        3. Construct a new node with a frequency equal to the sum of the frequencies of the two (2) removed nodes

        4. Connect the three (3) nodes, with the newly constructed node being the parent and the two (2) removed nodes being the children (the smallest should be on the left)

        5. If the new node is smaller than all other nodes or no nodes exist in the linked list

a. Add the new node to the beginning of the linked list

6. Else find the first node in the linked list that is larger than the new node and add the new node before it

f. Construct an encoding table

g. Create compressed file

i. Open the input file

ii. If the output file exists

1. Delete the output file

iii. Open the output file

iv. Read a character from the input file

v. While not at the end of the input file

1. Find assigned Huffman code for the read character

2. For each bit in the code

a. If a full byte has been constructed

i. Write byte to output file

ii. Reset byte and counter

b. If the current bit in the code is a one (1)

i. Modify the bit corresponding to counter-designated place in the byte to be a one (1)

ii. Increment the counter

c. Else increment the counter

3. Read next character from the input file

vi. Write final byte to the output file

vii. Close the output file

h. If key file exists

i. Delete the key file

i. Open the key file

i. For each character frequency object in the character frequency array

1. If the frequency of the object is not zero (0)

a. Write a string representation of the character frequency object to a new line in the key file

j. Close the key file

6. Else (assumed to be decompression action route)

a. If the key file does not exist

i. Display missing key file error message

ii. Exit

b. Open the key file

i. Instantiate counter for total characters

ii. Read the first line from the key file

iii. While the read line is not null

1. Check read line for matching string representation of a character frequency object

2. Convert matching string representation to character frequency objects if found

3. Add the found frequency to the total characters counter

4. Read the next line

c. Close the key file

d. Initialize a linked list of nodes of type character frequency

    i. For each element in the array of character frequency objects

        1. If the frequency of the character frequency object is not zero (0)

            a. Initialize a node with the character frequency object inserted as the root of the tree

            b. Add the newly created node to the end of the linked list

e. Construct a binary tree of the character frequency objects

    i. While the count of linked list nodes is less than one (1)

        1. If only two nodes remain in the linked list and the first node is larger than the last

            a. Swap the first and last nodes

        2. Remove the first two (2) nodes from the linked list

        3. Construct a new node with a frequency equal to the sum of the frequencies of the two (2) removed nodes

        4. Connect the three (3) nodes, with the newly constructed node being the parent and the two (2) removed nodes being the children (the smallest should be on the left)

        5. If the new node is smaller than all other nodes or no nodes exist in the linked list

a. Add the new node to the beginning of the linked list

6. Else find the first node in the linked list that is larger than the new node and add the new node before it

f. If the output file already exists

i. Delete the output file

g. Open the input and output file

i. Read a character from the input file

ii. Instantiate counter and array to measure out full bytes

iii. While input file data is valid

1. If a full byte has been measured out

a. Search the binary tree for leaf nodes corresponding to path designated by bits

i. If bit is a one, traverse the tree to the left

ii. If bit is a zero, traverse the tree to the right

iii. If a leaf node is found and the total remaining character to print are greater than zero

1. Write the character corresponding to the leaf node to the output file

2. Return to the root of the tree

3. Decrease the remaining amount of characters in file

b. Reset the counter and byte measuring array

2. Else
    a. Add bit to the byte measuring array
    b. Increment counter
    c. Read the next character from the input file

iv. Close the input and output file

7. Exit

Data

Input File

The input file will either be a standard ASCII text file or a compressed binary bit text file depending on whether compression or decompression is the current action.

For compression, it will be assumed that the file is an ASCII text file and will be read a single character at a time in sequential order. If a file other than an ASCII text file is used, the results will be undefined.

For decompression, it will be assumed that the file was compressed using this program and a key file already exists. If no key file is found, the program will reject the process. The input file will be read bit by bit looking for ones (1) and zeros (0). No other characters or information will be recognized. If a file other than one compressed by this program is used, the results will be undefined.

Output File

The output file for compression will be a series of ones (1) and zeros (0) as the output is a binary representation of the original data. The output of the decompression process will be the original file assuming that the key file was not corrupted or tampered with.

Data Structures

An array containing 256 character frequency objects will be used in both compression and decompression to determine the frequency of the characters found in the original file. The decompression process will rely on a key file to gather the frequencies of the characters in the original file. Regular expression will be used to decipher the key file in this process. Access to the array is handled by perfectly hashing the ASCII value of the character to the index of the

array. After all frequencies have been determined, the array is sorted using the built-in sort method and is no longer able to perform flawless O(1) operations.

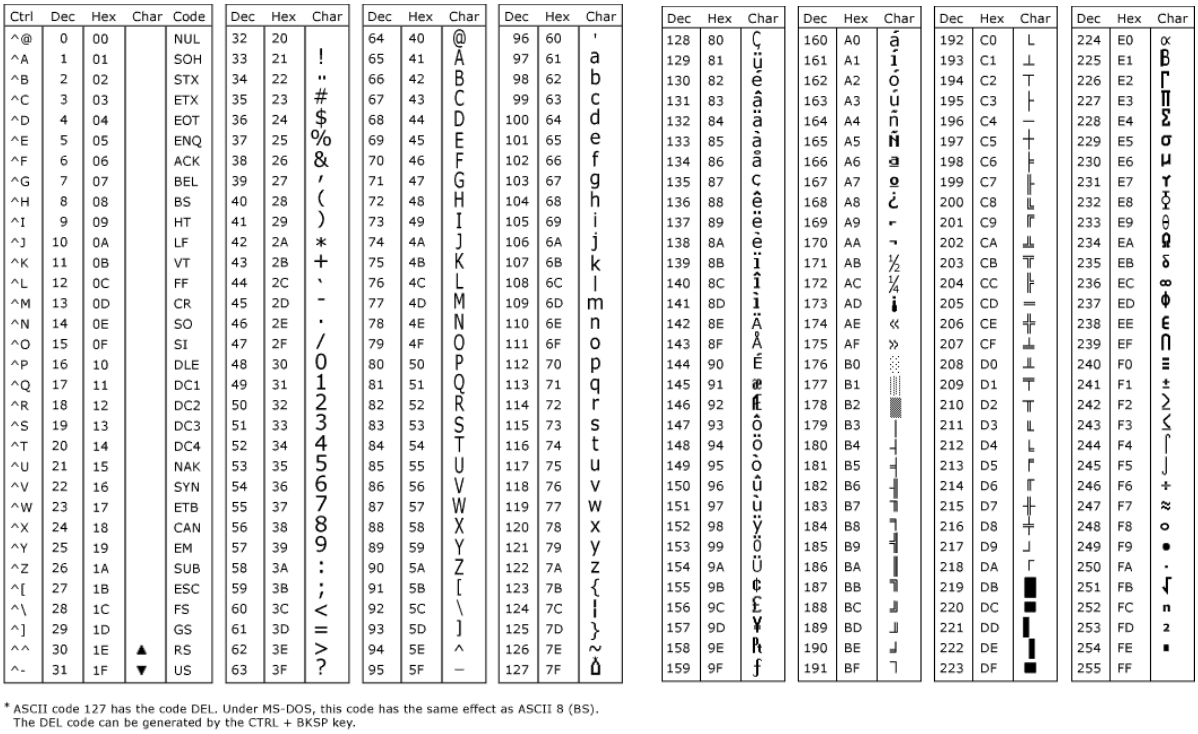
The sorted array is instantiated into a linked list of all character frequency objects with greater than zero occurrences registered. The linked list is used allow for quick removal of low frequency nodes to create a binary tree. A linked list is used rather than a stack so that composite nodes (nodes with a summed frequency of two smaller nodes) are still able to be re-inserted into the linked list in ascending order. This reinsertion is an O(n) operation that later allows for quick access through the binary tree.

A binary search tree is the primary data structure used to perform the Huffman coding process. The binary search tree branches with smaller nodes on the left and larger nodes on the right. This allows for the Huffman code designation by assigning a one or zero respectively to the traversal.

Footnotes

[1]Note: For a visible representation of the ASCII table, see Figure 1. Since some ASCII characters are invisible, they are identified as a string in the table, e.g. “NUL”, “SOH”, “BEL”, and “CR”.

Tables and Figures



Ctrl	Dec	Hex	Char	Code
^@	0	00		NUL
^A	1	01		SOH
^B	2	02		STX
^C	3	03		ETX
^D	4	04		EOT
^E	5	05		ENQ
^F	6	06		ACK
^G	7	07		BEL
^H	8	08		BS
^I	9	09		HT
^J	10	0A		LF
^K	11	0B		VT
^L	12	0C		FF
^M	13	0D		CR
^N	14	0E		SO
^O	15	0F		SI
^P	16	10		DLE
^Q	17	11		DC1
^R	18	12		DC2
^S	19	13		DC3
^T	20	14		DC4
^U	21	15		NAK
^V	22	16		SYN
^W	23	17		ETB
^X	24	18		CAN
^Y	25	19		EM
^Z	26	1A		SUB
^[	27	1B		ESC
^\	28	1C		FS
^]	29	1D		GS
^^	30	1E	▲	RS
^-	31	1F	▼	US

Dec	Hex	Char
32	20	
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?

Dec	Hex	Char
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_

Dec	Hex	Char
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	⌂

Dec	Hex	Char
128	80	Ç
129	81	ü
130	82	é
131	83	â
132	84	ä
133	85	à
134	86	å
135	87	ç
136	88	ê
137	89	ë
138	8A	è
139	8B	ï
140	8C	î
141	8D	ì
142	8E	Ä
143	8F	Å
144	90	É
145	91	æ
146	92	Æ
147	93	ô
148	94	ö
149	95	ò
150	96	û
151	97	ù
152	98	ÿ
153	99	Ö
154	9A	Ü
155	9B	¢
156	9C	£
157	9D	¥
158	9E	₧
159	9F	ƒ

Dec	Hex	Char
160	A0	á
161	A1	í
162	A2	ó
163	A3	ú
164	A4	ñ
165	A5	Ñ
166	A6	ª
167	A7	º
168	A8	¿
169	A9	⌐
170	AA	¬
171	AB	½
172	AC	¼
173	AD	¡
174	AE	«
175	AF	»
176	B0	░
177	B1	▒
178	B2	▓
179	B3	│
180	B4	┤
181	B5	╡
182	B6	╢
183	B7	╖
184	B8	╕
185	B9	╣
186	BA	║
187	BB	╗
188	BC	╝
189	BD	╜
190	BE	╛
191	BF	┐

Dec	Hex	Char
192	C0	└
193	C1	┴
194	C2	┬
195	C3	├
196	C4	─
197	C5	┼
198	C6	╞
199	C7	╟
200	C8	╚
201	C9	╔
202	CA	╩
203	CB	╦
204	CC	╠
205	CD	═
206	CE	╬
207	CF	╧
208	D0	╨
209	D1	╤
210	D2	╥
211	D3	╙
212	D4	╘
213	D5	╒
214	D6	╓
215	D7	╫
216	D8	╪
217	D9	┘
218	DA	┌
219	DB	█
220	DC	▄
221	DD	▌
222	DE	▐
223	DF	▀

Dec	Hex	Char
224	E0	α
225	E1	ß
226	E2	Γ
227	E3	π
228	E4	Σ
229	E5	σ
230	E6	µ
231	E7	τ
232	E8	Φ
233	E9	Θ
234	EA	Ω
235	EB	δ
236	EC	∞
237	ED	φ
238	EE	ε
239	EF	∩
240	F0	≡
241	F1	±
242	F2	≥
243	F3	≤
244	F4	⌠
245	F5	⌡
246	F6	÷
247	F7	≈
248	F8	°
249	F9	∙
250	FA	·
251	FB	√
252	FC	n
253	FD	2
254	FE	■
255	FF	

* ASCII code 127 has the code DEL. Under MS-DOS, this code has the same effect as ASCII 8 (BS). The DEL code can be generated by the CTRL + BKSP key.

Figure 1. The American Standard Code for Information Interchange (ASCII) table. The above images contain the decimal and hexadecimal values and their corresponding character. The charcters above 128 in decimal are known as the extended character set.