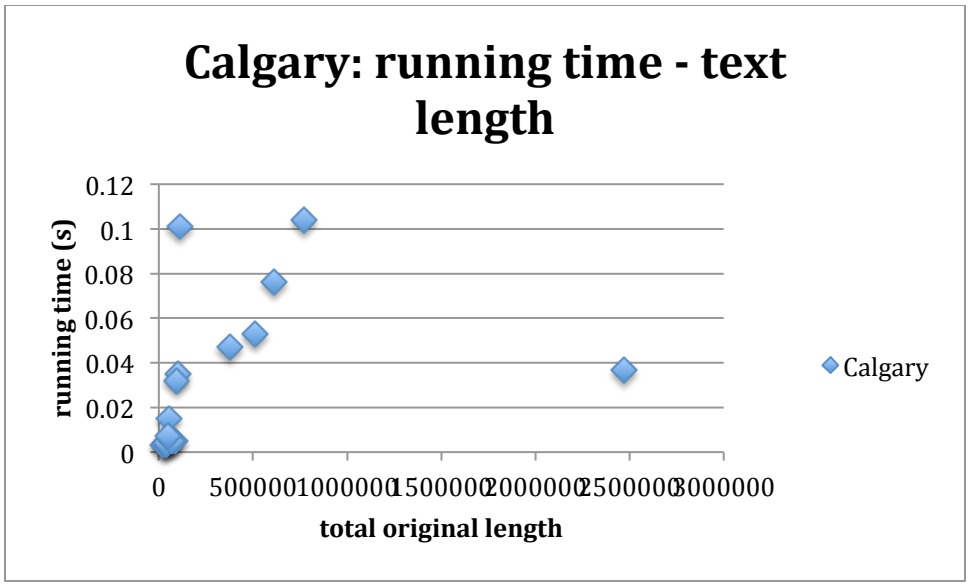
Huffman Analysis

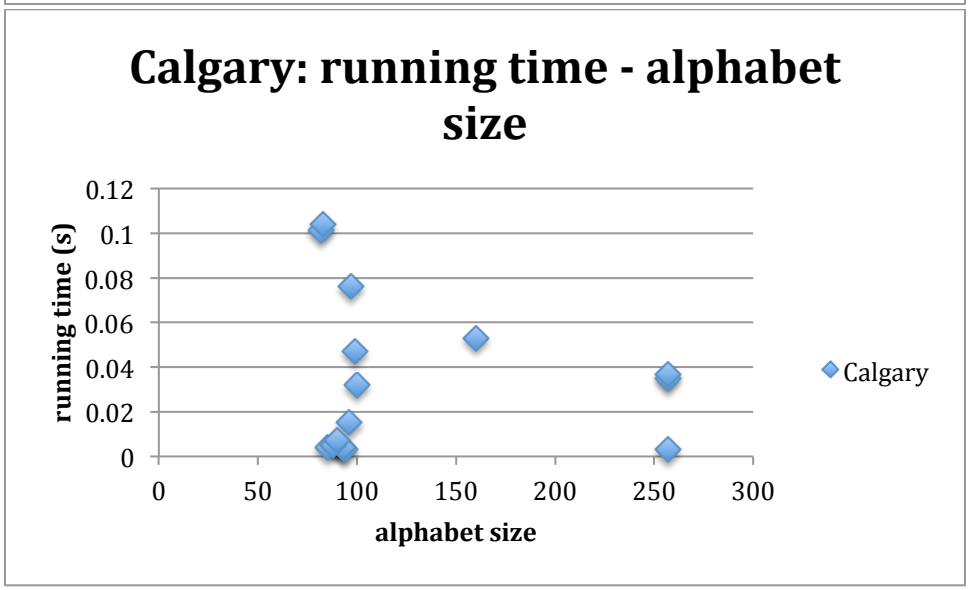
1. Benchmark your code on the given calgary and waterloo directories. Develop a hypothesis from your code and empirical data for how the *compression rate* and *time* depend on *file length* and *alphabet size*. Note that you will have to add a line or two of code to determine the size of the alphabet.

   Solution:
   For the Calgary file, run the "test" and get the following data

Calgary					
file	total original length	total new length	percent space saved(%)	alphabet size	running time(s)
bib	111261	72880	34.5	82	0.101
book1	768771	438495	41.89	83	0.104
book2	610856	368440	40.99	97	0.076
geo	102400	72917	40.2	257	0.035
news	377109	246536	39.14	99	0.047
obj1	21504	16411	38.97	257	0.003
obj2	2468174	194456	37.01	257	0.037
paper1	53161	33475	37.01	96	0.015
paper2	82199	47748	37.18	92	0.005
paper3	46526	27398	37.26	85	0.004
paper6	38105	24158	37.25	94	0.003
pic	513216	106778	44.49	160	0.053
progc	39611	26048	44.36	93	0.004
progl	71646	43110	44.25	88	0.005
progp	49379	30344	44.16	90	0.007
trans	93695	65361	43.76	100	0.032

   Plot the "running time" to "original length" and "running time" to "alphabet size"


Calgary: running time - text length
running time (s)
0.12
0.1
0.08
0.06
0.04
0.02
0
0
500000
1000000
1500000
2000000
2500000
3000000
total original length
Calgary


Calgary: running time - alphabet size
running time (s)
0.12
0.1
0.08
0.06
0.04
0.02
0
0
50
100
150
200
250
300
alphabet size
Calgary

Analysis:

waterloo					
file	total original length	total new length	percent space saved(%)	alphabet size	running time(s)
barb	262274	245919	6.24	231	0.105
bird	65666	56136	7.89	156	0.032
boat	262274	234733	9.05	231	0.05
bridge	65666	63542	8.47	257	0.042
camera	65666	58257	8.75	254	0.011
circles	65666	15902	14.32	21	0.004
clegg	2149096	2033920	7.76	257	0.18
crosses	657666	8525	9.5	19	0.004
france	333442	263360	10.65	250	0.017
frog	309388	193605	12.92	117	0.016
frymire	3706306	2187821	27.05	186	0.176
goldhill	65666	61886	26.87	227	0.006

From the graphs above, we can conclude that the running time ~ O((total original length)^2); while the running time is not clear how to relate to alphabet size.
On the other hand, analyzing the code, the running time ~ O(total original length); and running time (log(alphabet size)).

For the waterloo file, and get the following data (part of all)

Analysis similar to Calgary.

2. Do text files or binary (image) files compress more (compare the calgary (text) and waterloo (image) folders)? Explain why.

   Solution: comparing the "saved percentage", the compression of text files are much more efficient than compressing pictures. Since images are already saved as more efficient data structure – binary than text files, the compression of image files end into less efficient.

3. How much additional compression can be achieved by compressing an already compressed file? Explain why Huffman coding is or is not effective after the first compression.

   Solution: compress the file "kjv10.text" and then compress "kjv10.text.hf" to get the following data:

file	total original length	total new length	percent space saved(%)	alphabet size	running time(s)
kjv10.text	4345020	2488864	42.72	90	0.387
kjv10.text.hf	2488864	2451324	1.51	257	0.228

Analysis: from the above data, we know that the second compression is not effective at all with only 1.51% saved percentage while the first time is 42.72% saved percentage. After the first compression, the data are more compressed, which ended with bigger "alphabet size", which ended with low efficiency.

4. Devise another way to store the header so that the Huffman tree can be recreated (note: you do not have to store the tree directly, just whatever information you need to build the same tree again).

Solution: one method is directly store the 2-D string array of "HuffcodeArray", which store the character and corresponding Huffman code.