**Mark Eatough**

**CSIS 2430 9:00 Class**

**Programming Project 11**

**Huffman Coding Program**

**Assignment objective:**

Implement Huffman Coding - generate a file with 4000 Characters using Ipsum Lorum.  Convert to binary and measure the size.  Then, implement Huffman Coding to compress the file and then measure again.  What is the delta shrinkage from the first file to the second.  Now, use the frequencies found on page 771 #27 and reimplement.  Measure again.  Compare/contrast.

**What Worked?:**

I used a dictionary with letters as the key, and binary numbers as the value. I generated an Ipsum Lorum from <http://www.procato.com/lipsum>. The log base 2 of 26 is between 4 and 5 so for the constant binary strings I used 5 characters. I built a method that started at “00000” and then incremented the binary numbers from there. For the values found out of the book and the frequencies I came up with using the percentages of the Ipsum Lorum I imported I followed the example in the Huffman coding video example. I added the two smallest numbers in the dictionary and concatenated the strings until I had only two values. Each string that was added to another string was then added to a stack. Then I assigned a 0 as the value to one of the strings in the dictionary, and a 1 to the other. Then I popped the values off of the stack splitting the dictionary at those values as I went to create my new binary strings. This method seemed to work as doing this with the frequencies out of the book was more efficient than the constant 5 digit binary strings, and doing this with the frequencies I came up with was the most efficient.

**What did not work?:**

I originally tried to use a binary tree like was suggested in some of the videos, but I could not get this to work for me. Later, when using the dictionary to do the Huffman coding, I became quite concerned when some of my binary lengths were as 13 characters, which is obviously much larger than the 5 characters used for the constant lengths. When I saw this I tried to use my increment binary method I had used with the constants once the length was equal to 5, but this failed because it violated the no string can be the start of another string principal, so I eventually had to scrap that idea. My frequency binary strings were still shorter than my constant binary string in the end though, so it still worked as the very long strings were seldom used if at all.

**Comments:**

This was an interesting assignment to show how Huffman coding works. Using the frequencies out of the book made the file 13.44% smaller than using the constant size, and using the frequencies I generated made the file 19.29% smaller than the constants, and 6.76% smaller than the book frequencies. This was a good demonstration of why Huffman coding works. From the example I see that even if we use values that give a good estimation of how often a character will be used we can save quite a bit of space, giving us the same binary strings for all letters all of the time which could make compression software simpler to write and maintain. However, generating the frequencies based on how often characters show up in a specific file saves even more space.

1 '''  
 2 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*   
 3 \* Discrete Structures  
 4 \* Huffman Coding Program  
 5 \* Programmer: Mark Eatough  
 6 \* Course: CSIS 2430   
 7 \* Created Novermber 10, 2013  
 8   
 9 \*This program implements Huffman Coding, first we generate   
 10 \*file with 4000 Characters using Ipsum Lorum. We convert the  
 11 \*file to binary and measure the size. Then, implement   
 12 \*Huffman Coding to compress the file and then measure again.   
 13 \*What is the delta shrinkage from the first file to the second.   
 14 \*Now, use the frequencies found on page 771 #27 and reimplement.   
 15 \*Measure again. Compare/contrast.  
 16 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 17 '''  
 18 import os   
 19   
 20 #Ipsum Lorum generated using http://www.procato.com/lipsum/  
 21 #spaces removed and text shortened to 4000 characters using   
 22 #wordpad  
 23   
 24 #import ipusm lorm as read only   
 25 f = open('ipsumLorum.txt', 'r')   
 26 #assign file to variable  
 27 ipsumLorum = f.read()  
 28 #create dictionary to store letters with frequencies  
 29 frequencies = {}  
 30 #create dictionary to store letters with constant binary lengths  
 31 constant = {}  
 32 #create dictionary to store letters with book defined frequencies  
 33 book = {}  
 34 #print out ipsum lorum text  
 35 print ipsumLorum  
 36 #variables to keep track of first and last letter of alphabet  
 37 a = ord('a')  
 38 z = ord('z')  
 39 #method to increment the binary string  
 40 def incrementBinaryString(s):  
 41 return '{:05b}'.format(1 + int(s, 2))  
 42 #method to convert frequencies to binary strings  
 43 def dictFunction(dictionary):  
 44 #print dictionary  
 45 myStack = []  
 46 while(len(dictionary) > 2):  
 47 lowValue=1  
 48 lowKey="\*"  
 49 lowestValue = lowValue  
 50 lowestKey = lowKey  
 51 for key, value in dictionary.iteritems():  
 52 if(value < lowValue):  
 53 lowValue=value  
 54 lowKey=key  
 55 if(lowValue < lowestValue):  
 56 tempValue=lowestValue  
 57 tempKey=lowestKey  
 58 lowestValue=lowValue  
 59 lowestKey=lowKey  
 60 lowValue=tempValue  
 61 lowKey=tempKey  
 62 newValue = lowestValue+lowValue  
 63 newKey = lowestKey+lowKey  
 64 myStack.append(lowKey)  
 65 dictionary[newKey] = newValue  
 66 del dictionary[lowKey]  
 67 del dictionary[lowestKey]  
 68 for key in dictionary.iterkeys():  
 69 if(key == newKey):  
 70 dictionary[key]="1"  
 71 else:  
 72 dictionary[key]="0"  
 73 while(len(myStack)>0):  
 74 removedKey = myStack.pop()  
 75 for key, value in dictionary.iteritems():  
 76 if(removedKey[0] in key):   
 77 tempKey = key.split(removedKey)  
 78 dictionary[tempKey[0]]=value+"0"  
 79 dictionary[removedKey]=value+"1"  
 80 del dictionary[key]  
 81 break  
 82 print "\n\n\n", dictionary  
 83 #add all letters, and their occurances to a dictionary  
 84 #dictionary based on frequency  
 85 for character in range(a, z+1):  
 86 i = 0  
 87 for l in range(len(ipsumLorum)):  
 88 if(chr(character) == ipsumLorum[l]):  
 89 i+=1  
 90 frequencies[chr(character)] = float(i)/float(4000)  
 91 #convert ipsum lorum to binary digits and output to file  
 92 def toBinary(dictionary, s):  
 93 bin = ""  
 94 for letter in ipsumLorum:  
 95 for key, value in dictionary.iteritems():  
 96 if(letter == key):  
 97 bin+=value  
 98 break  
 99 g=open(s, "a")  
100 g.write(bin)  
101 g.close()  
102   
103 #add all letters, and constant binary strings to a dictionary  
104 #dictionary based on frequency  
105 x = '00000'   
106 for character in range(a,z+1):  
107 constant[chr(character)] = x  
108 x = incrementBinaryString(x)   
109 #constant = sorted([(value,key) for (key,value) in constant.items()], reverse=True)  
110 print "\n\n\nConstants:\n\n\n", constant  
111   
112 def deltaShrinkage(c, b, f):  
113 d1 = c-b  
114 s1 = float(d1)/float(c)  
115 print "\nThe delta shrinkage from the constant binary numbers to the book binary number is: %0.2f"%(s1\*100), "%"  
116 d2 = c-f  
117 s2 = float(d2)/float(c)  
118 print "\nThe delta shrinkage from the constant binary numbers to the frequency binary number is: %0.2f"%(s2\*100), "%"  
119 d3 = b-f  
120 s3 = float(d3)/float(b)  
121 print "\nThe delta shrinkage from the book binary numbers to the frequency binary number is: %0.2f"%(s3\*100), "%"  
122 #add all letters and frequencies out of book  
123 book['a'] = 0.0817  
124 book['b'] = 0.0145  
125 book['c'] = 0.0248  
126 book['d'] = 0.0431  
127 book['e'] = 0.1232  
128 book['f'] = 0.0209  
129 book['g'] = 0.0182  
130 book['h'] = 0.0668  
131 book['i'] = 0.0689  
132 book['j'] = 0.0010  
133 book['k'] = 0.0080  
134 book['l'] = 0.0397  
135 book['m'] = 0.0277  
136 book['n'] = 0.0662  
137 book['o'] = 0.0781  
138 book['p'] = 0.0156  
139 book['q'] = 0.0009  
140 book['r'] = 0.0572  
141 book['s'] = 0.0628  
142 book['t'] = 0.0905  
143 book['u'] = 0.0304  
144 book['v'] = 0.0102  
145 book['w'] = 0.0264  
146 book['x'] = 0.0015  
147 book['y'] = 0.0211  
148 book['z'] = 0.0005  
149   
150 #book = sorted([(value,key) for (key,value) in book.items()], reverse=True)  
151 #print "\n\n\n", book  
152 print "\n\n\nFrequencies:"  
153 dictFunction(frequencies)  
154 print "\n\n\nBook:"  
155 dictFunction(book)  
156 toBinary(constant, "constant.txt")  
157 toBinary(frequencies, "frequencies.txt")  
158 toBinary(book, "book.txt")  
159   
160 constantSize = os.path.getsize("constant.txt")  
161 bookSize = os.path.getsize("book.txt")  
162 frequenciesSize = os.path.getsize("frequencies.txt")  
163   
164 print "\n\n\nSize of text file with constant binary lengths", constantSize  
165 print "\n\n\nSize of text file with book frequencies", bookSize  
166 print "\n\n\nSize of text file with calculated frequencies", frequenciesSize  
167 print"\n\n"  
168   
169 deltaShrinkage(constantSize, bookSize, frequenciesSize)

