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.

```
'''
************************************************************
 * Discrete Structures
 * Huffman Coding Program
 * Programmer: Mark Eatough
 * Course: CSIS 2430
 * Created Novermber 10, 2013

 *This program implements Huffman Coding, first we generate
 *file with 4000 Characters using Ipsum Lorum. We convert the
 *file 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.
 ************************************************************
'''
import os

#Ipsum Lorum generated using http://www.procato.com/lipsum/
#spaces removed and text shortened to 4000 characters using
#wordpad

#import ipusm lorm as read only
f = open('ipsumLorum.txt', 'r')
#assign file to variable
ipsumLorum = f.read()
#create dictionary to store letters with frequencies
frequencies = {}
#create dictionary to store letters with constant binary lengths
constant = {}
#create dictionary to store letters with book defined frequencies
book = {}
#print out ipsum lorum text
print ipsumLorum
#variables to keep track of first and last letter of alphabet
a = ord('a')
z = ord('z')
#method to increment the binary string
def incrementBinaryString(s):
```

```
    return '{:05b}'.format(1 + int(s, 2))
#method to convert frequencies to binary strings
def dictFunction(dictionary):
    #print dictionary
    myStack = []
    while(len(dictionary) > 2):
        lowValue=1
        lowKey="*"
        lowestValue = lowValue
        lowestKey = lowKey
        for key, value in dictionary.iteritems():
            if(value < lowValue):
                lowValue=value
                lowKey=key
            if(lowValue < lowestValue):
                tempValue=lowestValue
                tempKey=lowestKey
                lowestValue=lowValue
                lowestKey=lowKey
                lowValue=tempValue
                lowKey=tempKey
        newValue = lowestValue+lowValue
        newKey = lowestKey+lowKey
        myStack.append(lowKey)
        dictionary[newKey] = newValue
        del dictionary[lowKey]
        del dictionary[lowestKey]
    for key in dictionary.iterkeys():
        if(key == newKey):
            dictionary[key]="1"
        else:
            dictionary[key]="0"
    while(len(myStack)>0):
        removedKey = myStack.pop()
        for key, value in dictionary.iteritems():
            if(removedKey[0] in key):
                tempKey = key.split(removedKey)
                dictionary[tempKey[0]]=value+"0"
                dictionary[removedKey]=value+"1"
                del dictionary[key]
                break
    print "\n\n\n", dictionary
#add all letters, and their occurances to a dictionary
#dictionary based on frequency
for character in range(a, z+1):
    i = 0
    for l in range(len(ipsumLorum)):
```

```
      if(chr(character) == ipsumLorum[l]):
         i+=1
   frequencies[chr(character)] = float(i)/float(4000)
#convert ipsum lorum to binary digits and output to file
def toBinary(dictionary, s):
   bin = ""
   for letter in ipsumLorum:
      for key, value in dictionary.iteritems():
         if(letter == key):
            bin+=value
            break
   g=open(s, "a")
   g.write(bin)
   g.close()

#add all letters, and constant binary strings to a dictionary
#dictionary based on frequency
x = '00000'
for character in range(a,z+1):
   constant[chr(character)] = x
   x = incrementBinaryString(x)
#constant = sorted([(value,key) for (key,value) in constant.items()], reverse=True)
print "\n\n\nConstants:\n\n\n", constant

def deltaShrinkage(c, b, f):
   d1 = c-b
   s1 = float(d1)/float(c)
   print "\nThe delta shrinkage from the constant binary numbers to the book binary number is: %0.2f"%(s1*100), "%"
   d2 = c-f
   s2 = float(d2)/float(c)
   print "\nThe delta shrinkage from the constant binary numbers to the frequency binary number is: %0.2f"%(s2*100), "%"
   d3 = b-f
   s3 = float(d3)/float(b)
   print "\nThe delta shrinkage from the book binary numbers to the frequency binary number is: %0.2f"%(s3*100), "%"
#add all letters and frequencies out of book
book['a'] = 0.0817
book['b'] = 0.0145
book['c'] = 0.0248
book['d'] = 0.0431
book['e'] = 0.1232
book['f'] = 0.0209
book['g'] = 0.0182
```

```
book['h'] = 0.0668
book['i'] = 0.0689
book['j'] = 0.0010
book['k'] = 0.0080
book['l'] = 0.0397
book['m'] = 0.0277
book['n'] = 0.0662
book['o'] = 0.0781
book['p'] = 0.0156
book['q'] = 0.0009
book['r'] = 0.0572
book['s'] = 0.0628
book['t'] = 0.0905
book['u'] = 0.0304
book['v'] = 0.0102
book['w'] = 0.0264
book['x'] = 0.0015
book['y'] = 0.0211
book['z'] = 0.0005

#book = sorted([(value,key) for (key,value) in book.items()], reverse=True)
#print "\n\n\n", book
print "\n\n\nFrequencies:"
dictFunction(frequencies)
print "\n\n\nBook:"
dictFunction(book)
toBinary(constant, "constant.txt")
toBinary(frequencies, "frequencies.txt")
toBinary(book, "book.txt")

constantSize =  os.path.getsize("constant.txt")
bookSize =  os.path.getsize("book.txt")
frequenciesSize =  os.path.getsize("frequencies.txt")

print "\n\n\nSize of text file with constant binary lengths", constantSize
print "\n\n\nSize of text file with book frequencies", bookSize
print "\n\n\nSize of text file with calculated frequencies", frequenciesSize
print"\n\n"

deltaShrinkage(constantSize, bookSize, frequenciesSize)
```

jGRASP Wedge2

```
Constants:


{'a': '00000', 'c': '00010', 'b': '00001', 'e': '00100', 'd': '00011', 'g': '001
10', 'f': '00101', 'i': '01000', 'h': '00111', 'k': '01010', 'j': '01001', 'm':
'01100', 'l': '01011', 'o': '01110', 'n': '01101', 'q': '10000', 'p': '01111', '
s': '10010', 'r': '10001', 'u': '10100', 't': '10011', 'w': '10110', 'v': '10101
', 'y': '11000', 'x': '10111', 'z': '11001'}


Frequencies:


{'n': '1010', 's': '000', 'e': '011', 'a': '1101', 'c': '11100', 'b': '1100111',
 'd': '11000', 'g': '1100110', 'f': '11001011', 'i': '001', 'h': '1100100', 'k':
 '110010100000', 'j': '110010101', 'm': '0100', 'l': '1001', 'o': '11101', 'q':
 '100011', 'p': '10000', 'r': '0101', 'u': '1111', 't': '1011', 'w': '110010100l
', 'v': '100010', 'y': '110010100001', 'x': '11001010001', 'z': '1100101000001'}


Book:


{'t': '000', 'y': '00100', 'a': '1110', 'c': '00110', 'b': '010110', 'e': '011',
 'd': '11111', 'g': '110010', 'f': '110011', 'i': '1011', 'h': '1010', 'k': '001
0111', 'j': '001011010', 'm': '01010', 'l': '11110', 'o': '1101', 'n': '1001', '
q': '0010110111', 'p': '010111', 's': '1000', 'r': '0100', 'u': '11000', 'w': '0
0111', 'v': '001010', 'x': '00101100', 'z': '0010110110'}


Size of text file with constant binary lengths 20000


Size of text file with book frequencies 17313


Size of text file with calculated frequencies 16143


The delta shrinkage from the constant binary numbers to the book binary number i
s: 13.44 %

The delta shrinkage from the constant binary numbers to the frequency binary num
ber is: 19.29 %

The delta shrinkage from the book binary numbers to the frequency binary number
is: 6.76 %
```