Alexis Atianzar & Arun Ramakrishnan
CSE 100
10/31/16

PA3 Checkpoint Writeup

Testing checkpoint1.txt:

(a)

In checkpoint1.txt:
abcdabcdabcdabcdabcdabcdabcdabcdabcdabcd

-Created a temporary output file called "testoutfile.txt" to call with compress

```
./compress checkpoint1.txt testoutfile.txt
```

In testoutfile.txt (Description):

```
Line 98:   10                  (header)
Line 99:   10
Line 100: 10
Line 101: 10
….
Line 257:
11100100111001001110010011100100111001001110010011100100111001001110010011100100111001
00
```

(every other line is 0)

(b)
-Created a temporary output file called "testinfile.txt" to call with uncompress

```
./uncompress testoutfile.txt testinfile.txt
```

In testinfile.txt (Description):
abcdabcdabcdabcdabcdabcdabcdabcdabcdabcd

(c)
Testing by hand: For checkpoint1.txt, we counted the number of times each symbol appeared and created the leaf node for each symbol with that number (its frequency). Then, using the Huffman Algorithm, we drew the tree, combining the c (of freq = 10) and d leaves (of freq=10) (which summed to 20) and then combining the a (of freq = 10) and b (of freq = 10) leaves (which also

summed to 20). Since those two parents (of count 20) were the only ones left, we combine those two nodes into the root node, which summed to 40.

From there, we labeled the path to the left children as ‘1’ and the path to the right children ‘0’. This assignment system led to these encoded sequences for each symbol:

a = 11
b = 10
c = 01
d = 00

Then, we rewrote the original message in checkpoint1.txt using these encoded sequences in place of the symbols and compared it to the printed output after the header in testoutfile.txt after we called compress, and it printed the same. In terms of the header, we made sure the lines in the header of testoutfile.txt printed out the frequency of the corresponding ASCII character in the file (off by one). As expected, lines 98-101 printed out 10, since a-d have the ASCII values 97-100.
To make sure our encoding worked correctly, we called uncompress on testoutfile.txt, which ended up returning the same original message in checkpoint1.txt.

Testing checkpoint2.txt:

(a)

In checkpoint2.txt:
aabbbbccccccccddddddddddddddddddddddddddddddddddddddccccccccbbbbaa

-Used temporary output file called “testoutfile.txt” to call with compress

./compress checkpoint2.txt testoutfile.txt

In testoutfile.txt (Description):

```
Line 98:   4                         (header)
Line 99:   8
Line 100: 16
Line 101: 32
….
Line 257:
000000001001001001010101010101010101011111111111111111111111111111111111111110101010101
0101001001001001000000
```

(every other line is 0)

(b)
-Created a temporary output file called “testinfile.txt” to call with uncompress

```
./uncompress testoutfile.txt testinfile.txt
```

In testinfile.txt (Description):
aabbbbccccccccddddddddddddddddddddddddddddddddccccccccbbbbaa

(c)
Testing by hand: For checkpoint2.txt, we counted the number of times each symbol appeared and created the leaf node for each symbol with that number (its frequency). Then, using Huffman Algorithm, we drew the tree, combining the b (of freq = 8) and a (of freq = 4) leaves, since they have the lowest frequencies, into a node with summed count 12. Then we combined that combination node (of count 12) and the c leaf (of freq = 16) into another node, of summed count 28. And lastly, we combined that count 16 node with the d leaf (of freq = 32) into the root node, of summed count 60.

From there, we labeled the path to the left children as '1' and the path to the right children as '0'. This assignment system led to these encoded sequences for each symbol in the file:

d = 1
c = 01
b = 001
a = 000

Then, we rewrote the original message in checkpoint2.txt using these encoded sequences in place of the symbols and compared it to the printed output after the header in testoutfile.txt after we called compress, and it printed the same. In terms of the header, we made sure the lines in the header of testoutfile.txt printed out the frequency of the corresponding ASCII character in the file (off by one). As expected, lines 98-101 printed out 4,8,16,32, respectively, since a-d have the ASCII values 97-100.
To make sure our encoding worked correctly, we called uncompress on testoutfile.txt, which ended up returning the same original message in checkpoint2.txt.

Overall: We knew our programs were working if we compressed a file and then uncompressed it, resulting in the same original file.