4. (a)
String:
a brigade strokes the varied basis.

Encoded result:
10010001111110001101111111010011010111100011110101111000011001011101110111100010
111000111000010110010011100011011111010101000111110011111011011111011001011

The number of characters in the string is 35 , so it’s fixed length is : 35 characters * 8 = 280 bits. Now, the result generated from Huffman tree has 157 bits. 1 - (157 bits/ 280 bits)= 0.56 ~= 0.44 * 100 = 44%
So 44% fewer bits were used.

Lookup Table of inserted File:
{ =00} {f=010000} {!=010001} {w=01001} {m=01010}
{v=0101100} {x=010110100} {:=010110101} {-=01011011} {k=0101110}
{p=01011110} {"=01011111} {o=0110} {l=01110} {b=01111}
{h=1000} {a=1001} {y=101000} {c=101001} {d=10101}
{t=1011} {r=11000} {j=1100100} {;=1100101000} {?=1100101001}
{'=110010101} {.=11001011} {,=110011} {n=11010} {i=11011}
{e=1110} {s=11110} {g=111110} {u=111111}

Above is the (key,value) lookup table of the Jabberwock.txt file.

I feel that the encoded string matched the source text frequencies because of the following:

1- Comparing/tracing each character in the string “a brigade strokes the varied basis.” to it’s string bit representation in the lookup table confirms that the matching process is correct.
2 - A space ‘ ‘ character, which is expected to be frequently used, consists of only 2 bits “00”. A non frequently used character such as a question mark “?” consists of 10 bits “1100101001”. That indicates that the Huffman algorithm is working.

4. (b)
The output of tracing all operations from the Operations.txt file:

Traversal = 2 1 4 3 0 5 6 10 12 11 15 14 21 20 24 23 17 25 13 7 28 30 33 32 31 35 34 29 27 38 39 37 36 41 42 43 47 46 45 44 40 49 53 54 50 57 55 58 94 79 98……………………………………………….
68766 **compares**
5308 **Zig Zigs**
4840 **Zig Zags**

Below are some of the top counts of all operations in Operations.txt:

Element:679 ->16 operations.
Element:322 ->16 operations.
Element:785 ->16 operations.
Element:33 ->16 operations……..

Given that Splay trees reshape themselves based on recent lookups, I have learned that if I’d only need access to a specific subset of elements of a tree, or if I want access to specific elements much more frequently than others, then the splay tree will outperform the AVL tree. After checking that a subset of the elements in operations.txt file are accessed more frequently than the others, I feel convinced that this is indeed the right implementation. In the case that the elements won’t be as frequently accessed and will be randomly selected, an AVL tree implementation would be better since the depth of there tree and operations costs will be limited to exactly O(Log n). In that case, We won’t be able to benefit from the splay operation, as bringing the selected element or elements up to the root won’t serve us if the next m operations will have nothing to do with these elements.