Abstract

This is documentation on the Huffman encoding algorithm and the successful testing and running of the Huffman encoding and decoding program

Huffman encoding/decoding

Ben Annis

**Huffman Encoding/Decoding**

For this project we are required to create a compression program that is based off of the Huffman encoding algorithm. This algorithm will take our file and assign the more frequently occurring characters a smaller number of bits and for the less occurring characters, it will assign them a larger number of fixed bits. For this assignment the steps are as follows:

1. We will need to read a file into our program
   1. For this program we will be using the wap.txt file and be reading it in
2. Once we are able to read this file into the program we will then need to count up all of the characters and their frequencies that are in the file
3. Once we are here the easiest thing to do is going to be to sort all of the character frequencies from lowest to highest in tree nodes.
4. Once these are sorted we will need to add these frequency chars to a binary tree structure and then assign them a binary code
5. For this we will need to use a binary tree
6. We will look at the two smallest nodes and then we will combine them by summing their frequencies and then adding them back into the tree. For an example please view the image below.
7. We will continue to keep taking the smallest two nodes from the tree and adding them together and putting them back into the tree until we have one node left
8. That last element that we have in our binary tree acts as our root node.
9. When it comes time to generate our code we will return 0 every time we go left and 1 every time we go right when we traverse the tree.
10. At this time we this will be the encoded file, now we will want to write our output to a file
11. Along with the output we need to put in the file a header so that when we want to decode we can know how the tree needs to be rebuilt.
12. Now if we want to decode it we will need to use the file header so that we know what all of the 1’s and 0’s mean.
13. We use the decode method in our code to read an encoded input file.

So for example there was a question on the exam that we took regarding the Huffman tree so I figured I would re-use the character frequencies example (Thanks Brian) and my answer to the question to better explain what we are doing with our Huffman tree.

|  |  |
| --- | --- |
| Character | Frequency |
| A | 25 |
| B | 25 |
| C | 10 |
| D | 10 |
| E | 15 |

First we need to sort the frequencies in acending order so 10,10,15,25,25 then what we need to do is take the first two frequencies and combine them so we have 20 and then insert it back into the list so we have 15,20,25,25, then we continue to do the same thing. We then combine 15 and 20 so we get 35 and then add it back into the list so we have 25,25,35. We do this again and combine 25 and 25 and get 50 then add back into the list so we have 35,50. We do this one more time to get one last number and we get 85. So our table or tree would look like this (view the next page)

E = 00

C = 010

D = 011

A = 10

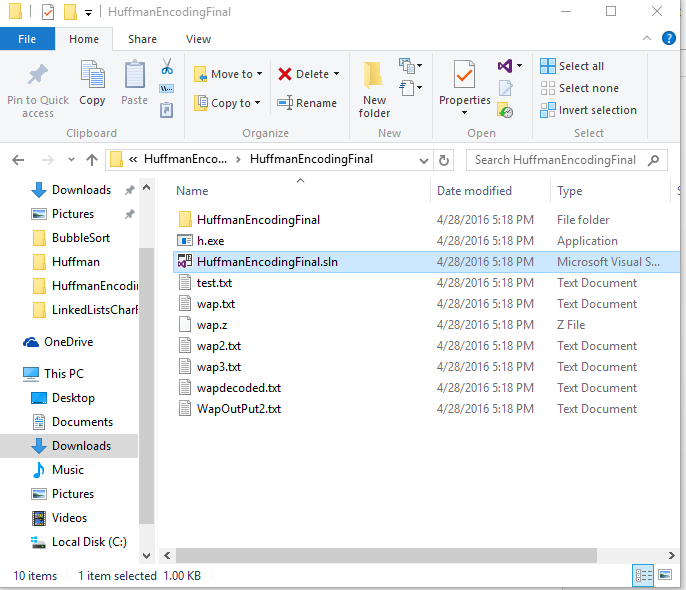
B = 11

To test the program view the testing section below.

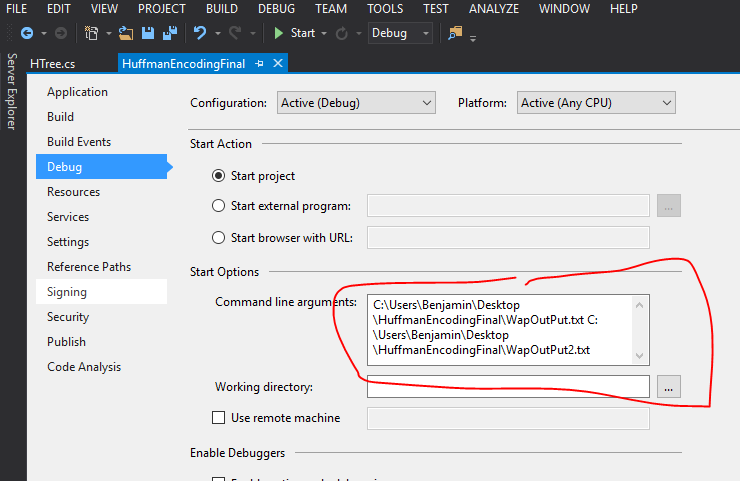
Testing the Code

**To test the code you can follow the examples that I used.**

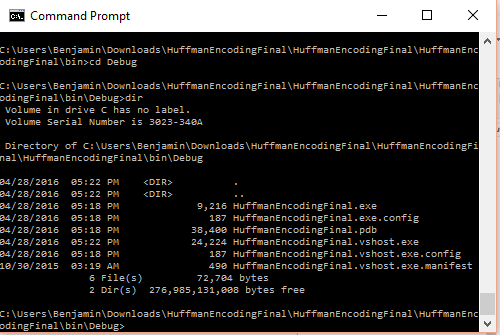
1. Extract and unzip the file so that it can be opened from a command line.



1. The first thing you will have to do is go into the program and navigate to the project properties. Here you will change the executable file paths.



1. Open the CMD in the windows start menu and navigate to the directory that the executable is located in. For me the path would be C:\Users\Benjamin\Downloads\HuffmanEncodingFinal\HuffmanEncodingFinal\HuffmanEncodingFinal\bin\Debug> and you would do this by using the Cd command to change the directory



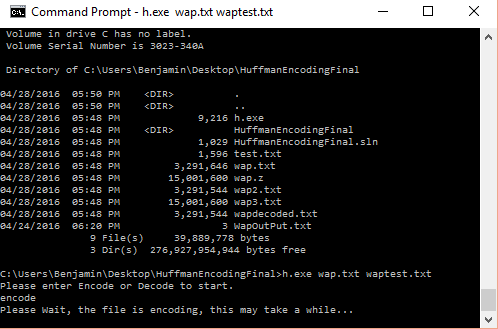
1. Once you get here you can run the executable which I have put into the folder called h.exe by using the command:

h.exe wap.txt wap(whatever you want the output to be called).txt

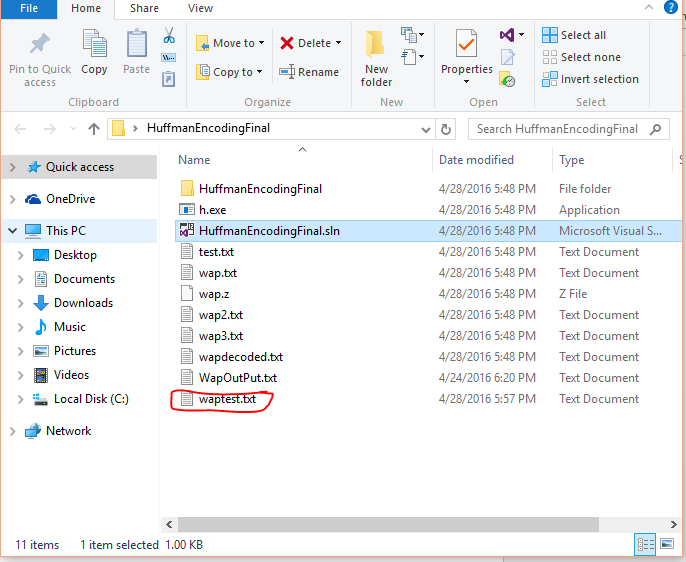
this program accepts two arguments, an input file and an output file.

Then hit enter.

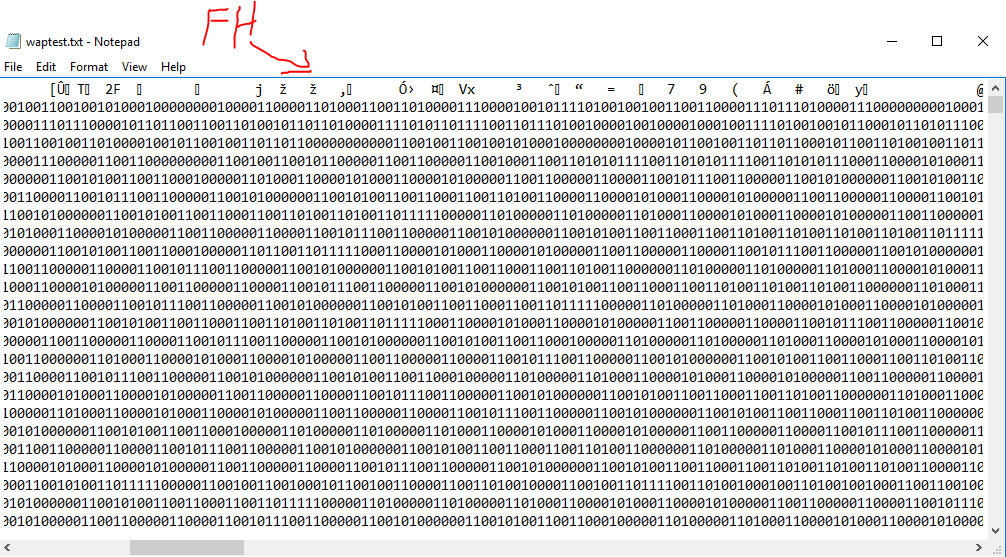
1. It should then ask you if you would like to encode or decode. My program at this time does not include error handling for this, so if you try to encode a file that is already encode, it is not going to work, it will crash. If you try to decode a regular text document then it will crash.



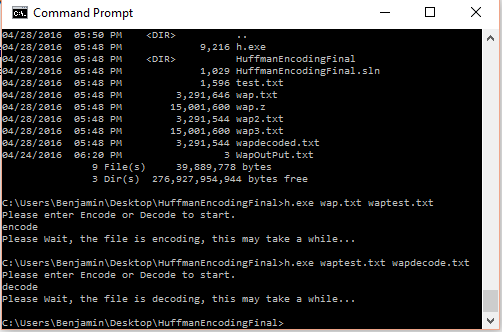
1. It may take some time but then it will print the output in the text document that you named



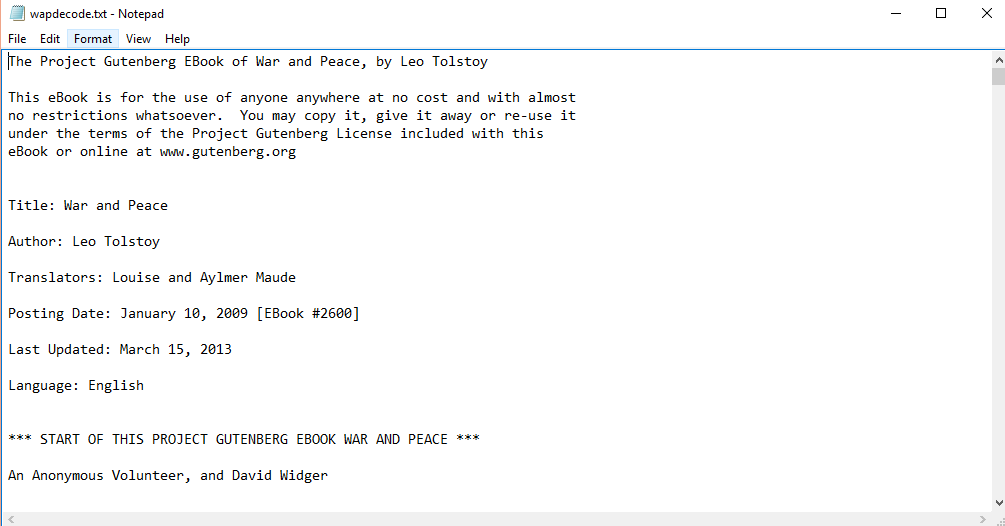
Then here are the contents of the file, as you can see at the top there is our binary output file header, this is what we will use to reconstruct the file.



1. The next step now is to decode
2. If you go back to the command line that we have open in steps 3 follow the same steps, to navigate back to the file directory, however when we go to decode our file we are going to use the decode command so it will look like: h.exe waptest.txt wapdecode.txt. enter decode and hit enter.



Now the output is back to a human readable text document.



**NOTE: This program does not shrink the number of bits per byte so it actually does not compress, however it still does use Huffman encoding/decoding.**

**Sources**

For my program I based the structure off some code that I found here: <http://snipd.net/huffman-coding-in-c> however I re-wrote the entire thing to make it my own work.