# Overview

### The purpose of this project is to design and implement a program that will compress and decompress an ASCII text file using a variation of the Huffman encoding algorithm. The program will function from the command line.

### INSTRUCTIONS

##### Run the program from the command line. The proper usage is:

##### ***Huffman.exe input.txt output.txt***

### The default input file is “War and Peace” and the name of the input file is “wap.txt” and the output file is “out.txt”, but they can both be overridden from the command line. The first command line parameter is assumed to be the input file name and path.

##### The deliverable of the project is the compressed file, encoding table and decompressed file. To access the compressed file, encoding table, and decompressed file, you will need to look for them in the bin-debug folder – the encoding table and the decompressed file will be in the form:

##### **“encoding” + outputFileName.txt**

##### **“decompressed” + inputFileName.txt**

##### **SCOPE**

##### The program will read in a text file from the command line.

##### It will read it character by character, counting the frequency of each character by using a linked list data structure.

##### Then, using a sorted linked list data structure, the program will add the character frequencies (from smallest to largest) to binary tree nodes. By combining the smallest two nodes and summing their frequencies, it will create a binary tree.

##### Using the “in-Order” tree traversal, each leg in the tree will be marked as “0” for left and “1” for right. The program will output the encoding table to a separate text file.

##### To compress the file, the program will read the file character by character, and it will find its corresponding encoding in the encoding table. The output is intended to be gibberish because each compressed byte will correspond to an ASCII value at chance.

##### To decompress the file that was just compressed, the program will read in the encoding table, and reconstruct the binary tree. Each time we get a leaf node, the program will write out to the file.

# PROCESSING

## Main Processing

### If the number of command line arguments is not 2

##### Display instructions of proper usage

##### Exit

##### Create an empty linked list of type character frequency

##### Read from the file

##### While not at the end of the input file

##### Create a linked list node of type character frequency and add it to the beginning of the list (find frequency method)

##### Create a check to find the characters

##### While check is false and node is not null

##### Compare the character frequency in the node to the character frequency

##### Increment the frequency

##### If character is not found, move to the next node in the list

##### If variable is true and character is not found in the list, add it to the end of the list

##### Read the next character

##### Start the search at the beginning of the list

##### While check is false and node is not null (find frequency method)

##### Compare the character frequency in the node to the character frequency

##### Increment the frequency

##### If character is not found, move to the next node in the list

##### Create a sorted linked list of type binary tree nodes of character frequencies to be able to sort the nodes and add them in order

##### For each character frequency in the list (that were read from the file)

##### Add a binary tree node (that takes in the character and its frequency) to the sorted linked list.

##### While the sorted linked list has more than one element (is not empty).

##### Create a **left** binary tree node of type character frequency.

##### The left node will take the value of the first element in the sorted linked list.

##### Remove the first element from the sorted linked list.

##### Create a **right** binary tree node of type character frequency.

##### The right node will now take the value of the first element in the sorted linked list.

##### Remove the first element from the sorted linked list.

##### Create a **place holder** binary tree node of type character frequency

##### The place holder node will take in the value null for data and the sum of the left and right node values as the value of the node.

##### The node with the smallest value (left node) now becomes the left node of the place holder node.

##### The node with the bigger value (right node) now becomes the right node of the place holder node.

##### Add the new node (with null data and the new value – sum of the two smallest nodes) to the sorted linked list.

##### Create the tree of type character frequency that takes in the first (and only value) of the sorted linked list.

##### Use the inOrder tree traversal and take in the root of the tree and the empty string for the encoding table, as parameters.

##### For each encodingData object in the encoding table

##### Write the encoding table to a separate file for later use.

## Compression

### Create a second output file. While we are not at the end of the input file:

##### Read from the file character by character.

* 1. Use the findEncoding method to find the character’s encoding in the encoding table (the method takes in the character and a linked list of encoding data).
     1. Create a linked list node of type encoding data and add it to the beginning of the list.
     2. Create a check to find the encoding and a string to hold it.
        1. While the encoding is not found and the node is not null
           1. If the value of the character in the encoding data is equal to the character that we are looking for:

Encoding is found

The string will now hold the encoding of the found character.

* + - * 1. Otherwise move to the next node.
      1. Return the encoding.
  1. For each character in the encoding string
     1. If the character is “1”
        1. Turn the bit on.
     2. Decrement the power variable by one (the power variable can be a maximum of 7).
     3. If the power variable is less than 0
        1. Reset its value to 7.
        2. Write the byte to a file in character form.
        3. Reset the byte to 0.

1. If the power variable is not equal to 7 (if we found our character but we are not at the end of the byte)
   1. Write the byte to a file in character form.

## Decompression

##### Create a new input file that reads the previously saved encoding file.

##### Create a binary tree with data and value 0.

##### Create a binary tree node that is the root of the tree.

##### While not at the end of the input file

##### Read the file line by line.

##### Create an array of strings to hold the values you get.

##### Split each line on the specified delimiter “»” (ASCII character 187).

##### If the array has more than one index

##### For each character in the string array:

##### If the character is “0”

##### And if the left node is null

##### Add a new binary tree node with data and value equal to 0, to the left.

##### Otherwise move to the left node.

##### Otherwise

##### If the right node is null

##### Add a new binary tree node with data and value equal to 0, to the right.

##### Otherwise move to the right node.

##### Declare a holding variable of type byte

##### Convert the string representation of the character from the encoding table (holder array index 0) into its byte equivalent.

##### Cast the byte equivalent into type char and add it to the node’s data.

##### Reset the node to be the root of the tree.

##### Create a new output and input file

##### The power variable is equal to 7, the node is the root of the tree

##### While not at the end of the input file (the compressed file)

##### Read the file in byte by byte

##### While the power variable is greater than or equal to 0

##### Turn the bit off

##### If the result is greater than 0 (if it’s 1)

##### Go to the right node

##### Otherwise, go to the left node

##### If the node is a leaf

##### Write the data to the file

##### Reset the node to be the root of the tree

##### Decrease the value of the power variable by one

##### The power variable is reset to 7

# COMPONENTS

## Character Frequency Class

##### UML

##### ***Properties***

##### Protect the data inside the class – In order to set the frequency, the value has to be greater than 0 (cannot have a negative counter).

##### ***Methods***

##### Increment() – increments the frequency by 1.

##### CompareTo() – overridden method, compares the frequencies of two objects.

##### If the current frequency is less than the frequency of the object we are comparing it to, returns -1.

##### If the current frequency is equal to the frequency of the object we are comparing it to, returns 0.

##### If the current frequency is greater than the frequency of the object we are comparing it to, returns 1.

##### Equals() – If the two objects have the same frequency and same character value, return true.

##### ToString() – overridden method.

## Binary Tree Class

##### UML

##### ***Properties***

##### Protect the data inside the class.

##### ***Methods***

##### inOrder() – Takes in a binary tree node of type characterFrequency, and an encoding string.

##### If the node is not null, it assigns 0 to the left branches and 1 to the right branches.

##### If the node is a leaf, adds the character and its encoding to the encoding table.

## Binary Tree Node Class

##### UML

##### ***Properties***

##### Protect the data inside the class.

##### ***Methods***

##### isLeaf() – The node is a leaf node if both the right and left sides are null.

##### CompareTo() - overridden method, compares the numerical values of two nodes.

##### If the current value is less than the value of the object we are comparing it to, returns -1.

##### If the current value is equal to the value of the object we are comparing it to, returns 0.

##### If the current value is greater than the value of the object we are comparing it to, returns 1.

## Encoding Data Class

##### UML

##### ***Properties***

##### Protect the data inside the class.

##### ***Methods***

##### ToString() – Overridden method. Formats the string in the following format, where “»” is the delimiter:

##### value » encoding

## Sorted Linked List Class

##### UML

##### ***Methods***

##### Add(element: E) – The method will check:

##### If the element entered is null – return nothing.

##### If the list is empty – add the element at the beginning of the list.

##### Otherwise, if the element is less than the value of the first node – add the element at the beginning of the list.

##### Otherwise, if the element is greater than the value of the last node – add the element at the end of the list.

##### Otherwise create a LinkedListNode to loop through each node and compare the value of the element to the value of each node.

##### If the element is less than the value of the node – add it before the node.

##### Otherwise – move to the next node in the list.

# Testing

## Compression/Decompression Testing

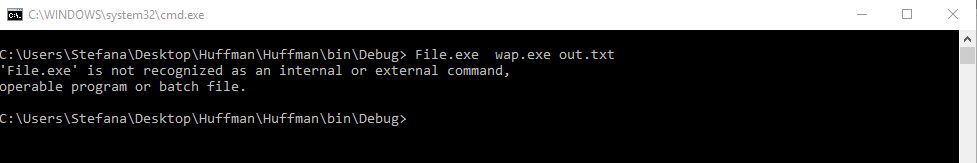
Since the compression and decompression are not two separate programs, there will only be one testing document.

The table shows an overview of the testing scenarios and the status of running scenarios.

|  |  |  |
| --- | --- | --- |
| Scenario | Description | Pass/Fail |
| 1 | Test that program can accept command line arguments and handle incorrect ones | PASS |
| 2 | Test for non-existent input file | PASS |
| 3 | Test for known input file | PASS |
| 4 | Test for number of command line arguments | PASS |

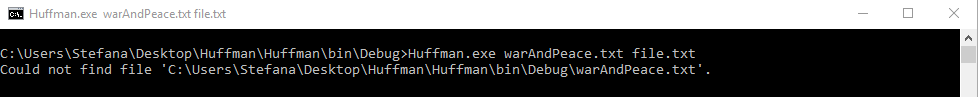
Scenario #1- Open the command window and type in the wrong program name, or wrong input, the program will display an error message.

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | From the command line, input the wrong file name (File.exe instead of Huffman.exe). | Error Message |
| EXPECTED OUTPUT | | Error Message |
| ACTUAL OUTPUT | | Error Message |
| RESULTS | | PASS |



Scenario #2- While running the program from the command line, input the wrong file name (the name of a non-existent file) and the program will display an error message.

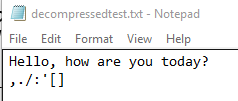
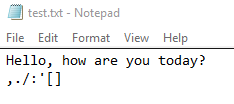
|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | Input parameter with wrong file name. | Huffman.exe warAndPeace.txt Out.txt |
| EXPECTED OUTPUT | | Error Message |
| ACTUAL OUTPUT | | Error Message |
| RESULTS | | PASS |



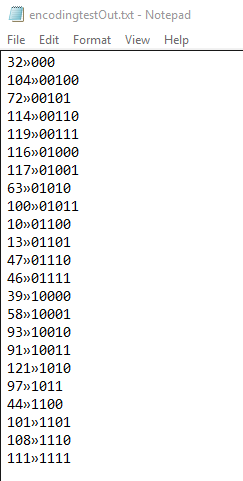
Scenario #3- While running the program from the command line, test with known input (example – Hello, how are you today? ,./:'[]), and the program will successfully compress and decompress the text. The file with the above text will be included in the deliverable.

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | Create a text file with the following text: Hello, how are you today? ,./:'[] | Output: Test.txt |
| 2 | Run the program from the command line, passing the file name as the input parameter. | Input: Huffman.exe Test.txt Out.txt |
| 3 | The program will output a compressed file with the name you specified (Out.txt in this scenario). | Output: Out.txt |
| 4 | The program will also output an encoding table. | Output: encodingOut.txt |
| 5 | The program will also output a decompressed file. | Output: decompressedOut.txt |
| EXPECTED OUTPUT | | Output a compressed file, an encoding table and a decompressed file. |
| ACTUAL OUTPUT | | Output a compressed file, an encoding table and a decompressed file. |
| RESULTS | | PASS |

##### C:\Users\Stefana\AppData\Local\Microsoft\Windows\INetCacheContent.Word\t3.png

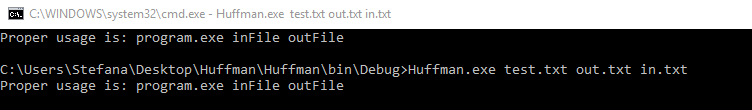


##### (encoding table)



Scenario #4- While running the program from the command line, test the program with an invalid number of arguments and the program will display an error message.

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | Input 3 command line arguments. | Huffman.exe test.txt out.txt in.txt |
| EXPECTED OUTPUT | | Error Message |
| ACTUAL OUTPUT | | Error Message |
| RESULTS | | PASS |



Scenario #5- Create an empty file and test that the compression/decompression works.

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | Create an empty file empty.txt. | Output: empty.txt |
| 2 | Run the program from the command line, passing the file name as the input parameter. | Input: Huffman.exe empty.txt Out.txt |
| 3 | The program will output an empty compressed file with the name you specified (Out.txt in this scenario). | Output: Out.txt |
| 4 | The program will also output an encoding table. It’s format can be seen below. | Output: encodingOut.txt |
| 5 | The program will also output an empty decompressed file. | Output: decompressedOut.txt |
| EXPECTED OUTPUT | | Output an empty compressed file, an encoding table and an empty decompressed file. |
| ACTUAL OUTPUT | | Output an empty compressed file, an encoding table and an empty decompressed file. |
| RESULTS | | PASS |

