Overview

The purpose of this program is to encode a string into a Huffman tree, encode the string, and then decode it again.

### PROCESSING LOGIC

The Main method of the Program class is executed first and creates a CharacterManager object.

The CharacterManager object does most of the work here. It takes a string from a file or from the test data in the program and splits it into an array. In its current testing implementation it takes the string located in Main(), turns it into a char array, and passes that to HandleInput(), which counts the characters and returns an array of CharacterFrequency objects. This array is then compacted so it is not a sparse array by the CompactArray() method. The resulting array, also a CharacterFrequency[] array, is passed to SortFrequencies(), which sorts the objects by their frequency in ascending order. It is then passed to MakeSortedNodeList(), which creates a LinkedList<Node<CharacterFrequency>>. This is used to build the binary tree.

The BuildBinaryTree() method is not complete, as I was unable to successfully insert a combined node into the linked list at the appropriate locations.

Any specific algorithms to be used should be stated or referenced.

Character counting algorithm in the HandleInpout() method of CharacterManager

Sparse array compacting algorithm in the CompactArray() method of CharacterManager

Binary tree algorithm in the BuildBinaryTree() method of CharacterManager

### DATA (INPUT/OUTPUT)

The program will accept a file to compress by supplying the file and extension. It will also run without a provided file, in which case it will run using the test data supplied in Main().

### COMPONENTS (SOURCE CODE NAMES, CLASSES, METHODS)

Program class- entry point. Contains a single CharacterManager object instantiation, depending on the kind of input given.

CharacterManager class – creates the CharacterFrequency objects used throughout the program’s execution.

FIELDS

// unused

private int \_leafCounter = 0;

// stores the current directory

private string CurrentDir { get; }

// array for CharacterFrequency objects

private CharacterFrequency[] \_characterFrequencyObjectArray;

// linked list used to hold sorted tree nodes

private LinkedList<Node<CharacterFrequency>> \_sortedNodeList;

// array to hold the CharacterFrequerncy[] array once it is compacted

private CharacterFrequency[] \_compactedArray;

// array to hold the sxorted sparse array during its creation and prior to being compacted

private CharacterFrequency[] \_sortedArray;

METHODS

ReadFile() // reads the file using StreamReader

ProcessString() // processes the provided string into a character array and then passes it into ProcessData()

ProcessFile() // sends the contents of the file through .toCharArray() and the passes it into ProcessData()

ProcessData() // passes a character array into CompactArray to remove any null entries and return the result into \_compactedArray. Passes \_compactedArray into SortFrequencies() for sorting of each CharacterFrequency object into ascending order by their frequency, returning \_sortedArray. Passes that result into MakeNodeList, returning \_sortedNodeList, a linked list of nodes.

Had I been able to go further, I would have then combined pairs of nodes (the first two in the list) into a single node, then added it into the linked list until I had a single node comprising by whole list, made of a binary tree containing the objects and their frequencies in an order I could use for encoding.

I didn’t get that far because I even after hours I could not get any insertion to work on the linked list.