Huffman Compression Project

The Huffman Compression algorithm takes a bitree that holds Pairs of characters and their frequencies. It then takes the smallest of the frequencies adds them together to form a new frequency and puts them back into the bitree. The last node being the root and the frequencies removed and added being the leaves.

For this project the program takes an input file and feeds it into a frequency table that counts the characters. It then feeds that character count to the Huffman tree header that builds the prefix for the bitree. The Huffman encoder creates the bit codes and endcodes the inputed file which is then outputted to a new file.

Header Files

Object.h

Class: Object (you gave us this)

Allows us to print out objects using a toString() method that's overloaded in every other printing class. Used in Pair, FrequencyTable, HuffmanTree and HuffmanEncoder.

Pair.h

Class: Pair (mostly given to us in class)

This class creates a character, frequency pair to be used with the FrequencyTable.h (produces arrays of pairs) HuffmanTree (uses those pairs to create tree nodes) and the HuffmanEncoder (reads the paris to generate bit codes) This class is used to carry data through each header

BiTree.h

Class: BiTree (you gave us this)

BiTree is a node template to represent our HuffmanTree which contains data pairs and left and right child variables. It's used in the HuffmanTree.h to create BiTree<Pair> nodes, Heap.h to store BiTree<Pair> pointers, TreePrinter.h to print the BiTree<Pair> and the encoder to assign codes

TreePrinter.h

Class: TreePrinter

This class just helps me format the printing of the BiTree as (frequency left subtree, right subtree) or (10(4(2) (2)) (6)) it's not pretty but I think this follows the example you've given.

It mainly just reads nodes made by the HuffmanTree.h and used in the main program.

FrequencyTable.h

Class: FrequencyTable

This class reads the input file, ignoring white space, converts all letters to lowercase, counts how many times each character appears in the file and then creates an array of pairs then finally sorts the pairs by frequency using the CountSort algorithm. Pairs should appear as (character, frequency) or as (a, 20)(b,5)(c,30) etc.

Heap.h

Class: Heap.h

The heap class just heapifys the BiTree<pair> node. It pops the 2 lowest frequencies and then pushes them back into the BiTree. This class will always take the two lowest frequencies because that's whats needed for the Huffman algorithm and its needed to build a minheap. This class is used in FrequencyTable.h to build the BiTree nodes. HuffmanTree pushes the BiTree<pair> into the heap and also pops the smallest frequencies. So the heap class is the merge process of HuffmanTree.h

HuffmanTree.h

Class: HuffmanTree

The HuffmanTree class uses the sorted pairs to build the Huffman tree. It creates a Bitree<pair> leaf for each character/frequency pair, pushes all leaves into a minheap. While more than one tree exists it will pop the two smallest frequency leaves, add them, then push them back into the heap. The last node(the node with the highest frequency) will be the root of the tree.

This class takes pairs arrays from the FrequencyTable header uses Heap.h to merge nodes. Uses the BiTree structure you gave us to make Bitrees. Is printed by the printer and is finally passed off to the Encoder.

HuffmanEncoder.h

Class: HuffmanEncoder

This class walks the HuffmanTree header and builds a code table (character -> bitstring) it reads the original file again and uses the code to convert text into bits and then finally saves the encoded file as an output file.

This header takes the tree root from HuffmanTree.h walks the BiTree<pair> nodes uses the pair keys to fill code arrays and reads the original input file to finally write an encoded output file.