John Abrams - jsa109 | Chris Zachariah - cvz2
4/3/2019
CS 214 - Systems Programming
Asst2 - fileCompressor

ReadMe:

Overall:

The program takes in command line arguments and runs the necessary functions on the files/directories that are given. The program does sends an error in the case that not enough or too many inputs are given or if the wrong type of input is given. If given the correct inputs, it then does the necessary tasks on the file/directories depending on which specific flags are given.

Specifics: Function Name => Worst Case Runtime

main() => O(n^2)

- takes in command line arguments and checks for the right number and right type of arguments
- 6 different scenarios exist within the function: -b | -c | -d | -R & -b | -R & -c | -R & -d
- the regular scenarios need a filename or a path to a file name for the functions to do its work
- the recursive scenarios require directories or paths to directories

runBFlag(),bflag(),updateRoot(),orderNodes(),makeTree(), printTree(),deleteTree() => O(n^2)

- all these functions work together under the main helper function runBFlag()
- uses structs, char arrays and file descriptors

Process:

1. Takes a filename/filePath as the input
2. Reads through a file and tokenizes all the words and delimiters and make a Linked List with frequencies
3. Orders all the nodes based on frequency from lowest to highest frequency
4. Takes the Linked List of ordered nodes and using the basic Huffman Tree Code algorithm, construct a tree (with higher frequencies on the top branches and vice versa)
5. Make the HuffmanCodebook in the directory that the fileCompressor.c was called by recursively iterating through the tree in-order and write the words and bitString into the newly created HuffmanCodebook file
6. Close and free all file descriptors or char arrays or structs used for the compression process

cFlag(),addExt(),compressFile() => O(n^2)

- all these functions work under the main helper function cFlag()
- uses structs, char arrays and file descriptors

Process:

1. Make a new file called <givenFileName>.hcz
2. Go through the file given and tokenize all the words and delimiters that are found
3. Compare the words/delimiters to the tokens in the HuffmanCodebook and once the correct token has been found, write the corresponding bitString to the new file that was created
4. Close and free all file descriptors or char arrays or structs used for the compression process

dFlag(),decompressFile(),makeDecompNodes() => O(n^2)

- all these functions work under the main helper function
- uses structs, char arrays and file descriptors

Process:

1. Check to make sure that the file given does, in fact, end in ".hcz"
2. Run through the HuffmanCodebook and make a Linked List with each node containing the token name of chars and the corresponding bitString
3. Make the new file for the decompression (just take off the ".hcz" from the file given)
4. Run through the file given and the Linked List and find the correct node with the corresponding token to write into the new file created
5. Close and free all file descriptors or char arrays or structs used for the compression process

recurFlag() => O(n^2)

- uses file descriptors

Process:

1. Check to make sure that the program can get into the given directory
2. Build the file path when going through each file
3. Once a file that can be read is found, check the modes and run the correct flag function by giving the function the file path or create the file needed
4. Once a directory is found, call the current function again to now read files within the subdirectory
5. Make sure to close the file descriptors once the function gets to the end