CS 214: Systems Programming
Assignment 1: File Compressor

Authors:

Jonathan Konopka

NETID: jk1549

RUID: 178005220

Nicholas Rytelewski

NETID: nr548

RUID: 197000952

Project Description:

The goal of this project was to create a program that would be able to create and use a "HuffmanCodebook" file, which could be used to perform compression and decompression operations on targeted files. Different flags may be used in the command line arguments to perform such operations (SEE BELOW). We created the huffman codes in our program with the min-heap data structure. A linked list was used to tokenize the contents of a called file and track the frequency of tokens, along with being used for miscellaneous purposes depending on the given flag.

Our program took either of the following command line flags :

1. **"-b" : BUILD** (./fileCompressor -b <filename>)
2. **"-c" : COMPRESS** (./fileCompressor -c <filename> HuffmanCodebook)
3. **"-d" : DECOMPRESS** (./fileCompressor -d <filename>.hcz HuffmanCodebook)
4. **"-R" : RECURSIVE** (./fileCompressor -R <flag> <filename>)
   Having -R present before any previous flags makes the operation require a directory path rather than a file path, as specified above. This flag will collect all files within the directory specified and perform the operation on each. "-b" would make a single HuffmanCodebook to decipher all files within the directory. "-c" would create .hcz files for each non-hcz and non-codebook file found in the directory. "-d" would create the original decoded version of any .hcz files in the directory.

Time and Space Analysis

We had two data structures in our project; a min-heap, and a linked list. In many cases throughout the program, elements within the linked list or min-heap needed to be searched for, for example, to retrieve the token or ascii character arrays. To search through our linked list, the time complexity would be O(n). Likewise, searching through our min-heap would have a time complexity of O(log(n)), comparable to that of a binary search tree since after our heap was set up, it was traversed as so.

Functions:

void dir_contents(char* rootpath, int flag_b, int flag_c, int flag_d, char del)
Called when the program is in recursive mode (-R is supplied). Recursively gets the paths of each file and directory in the directory provided in the command line argument. Depending on other flags set, operation will occur.

void tokenize(char* filename)
Makes tokens from the file to be inserted into the linked list. Escape characters space, tab, and newline are the delimiters we chose to include.

void linkedinsert(char *str)
Inserts token into the linked list if not currently present. Takes the string token as an argument, and if it is already present on the list, it increments the frequency of that token by 1.

void listprint()
Diagnostic function used to print out our linked list and see if it works properly. Not used in runtime and was just for our own debugging

void freelist(linked* head)
Frees memory allocated to the linked list. Does not return anything but does take the head of the linked list as an argument.

int countlist(linked* head, int print)
Counts the size of the given linked list and returns the size in the end. Requires the head of the linked list and the integer as an argument. This value was used when creating the min-heap / is coincidentally the size of the min-heap.

heapNode* newNode(char* token, int freq)
Makes a new node for the min-heap with the given token and frequency. Returns the heapNode pointer.

void swap(headNode a, heapNode** b)**

Can swap two given min-heap nodes when called, used in the min-heap construction. Returns nothing since it was given double pointers.

void minHeapify(heap* heap, int index)
Heapify function, where the nodes are sorted properly in the minheap. Requires the index and the heap pointer so it can be used recursively.

heapNode* extractMin(heap* heap)
Get the minimum value node for the heap. Takes the pointer to the heap and then returns the node found to be smallest. Used when creating the min-heap.

void insertMinHeap(heap* heap, heapNode* heapNode)
Insert the new node into the given min-heap. Returns nothing. Used when creating the min-heap.

void buildMinHeap(heap* heap)
Constructs the minheap, acting as a helper function called in the createAndBuildMinHeap. Takes the heap as an argument but does not return anything

heap* createAndBuildMinHeap(heap* heap, int size)
Makes the min-heap of a given size, where both the initial size and the overall capacity is set to "size". Allocates an array of structs to each node, created by the called newNode function. Also takes the heap in the argument and also returns one created

heapNode* buildHuffmanTree(heap* heap, int size)
Builds the huffman tree, intended to make it out of the minheap returned and set by the createAndBuildMinHeap function. Takes the size integer and heap as arguments, return the root node.

void writeCodebook()
Creates the HuffmanCodebook file and then writes each token along with its ascii value into the codebook. Called specifically when the -b flag is set.

void generateBytestrings(heapNode* node, char* ascii)
Takes the root of the huffman tree and ascii and generates the huffman code byte string. It returns nothing, it just recursively descends through the left and right nodes of the tree, and concatenates a bytestring with 0 when descending left and 1 when descending right.

void compress(char* filename, char* codebookname)

Compress any file using the -c flag, taking in the name of the file and the codebook built. First it takes the ascii and token values out of the codebook, then can call the helper function compressed_tokenize. Returns nothing

void compressed_tokenize(char* filename)
Creates the compressed file with the hcz extension. Uses the codebook information before and the opened file to assign each token found to its proper ascii. Takes the name of that file to be opened as an argument then returns nothing

char decompress(char* codebookname);
Decompress any files using the -d flag. This is a 3 part function with recreate() and traverse(). Like compress, this takes the ascii and token values out of the code book and sends them to recreate().

void recreate(char* token, char* ascii);
Recreate creates the min heap nodes every time it is called by traversing the heap from the ascii (0 = go left, 1 = go right) starting from the root, and makes the last character of the ascii's node contain the token.

void traverse(char* filename, char del);
After decompress is called, traverse goes through the file supplied and creates the original file (without the .hcz) given the codebook.

void freeheap(heapNode* node);
Function used to free all nodes from the heap at the end of the operation. Traverses the layers of the min-heap and free()'s each node, token, etc.