ECE 368 Project 2 – Milestone 1

Dwezil D'souza

For this project I will be using Huffman encoding to develop a compression program as well as a decompression program for text-file input.

For the compression program:

Character counts table

I will parse the input file (a plain text file) and keep a count of each character in the file. I will make a table (using an associative array implementation) of all the 256 characters in the ASCII character set and their counts. The counts of the characters that do not occur in the input file will be set to 0. I will also manually add a pseudo-EOF character to the table with a count of 1. There are now 257 entries in this table.

Huffman Tree

To build the Huffman tree, I will first create a forest of 1-node trees of all the entries in the character counts table that have a count above 0. Then I will proceed to use the algorithm outlined in the project specification to build a single, optimal tree using all the one-node trees in this forest.

Priority queue

For the process of selecting the 2 trees with the smallest count at each stage of the Huffman-tree building algorithm, I plan to use a priority queue data structure with 'least count' being the priority. This will be implemented using a binary tree 'heap'. Each node in the heap will have two fields – the 'data' field containing a pointer to the root node of each tree in the forest; and a 'key' field containing the count of that root node. After each add or remove operation, the heap will reshuffle itself so that the node with the lowest count becomes the root node.

Huffman Table

Once I have the optimal Huffman tree I will create a table (implemented as a associative array) of each character in the ASCII set and their new Huffman encoded representation. I am now ready to start encoding the input file.

Writing to output (compressed file)

First, I will write a header to the file, which will provide the information needed to decode the compressed file. The first 2 bytes of the header will be the header size(not including the first 2 bytes themselves). I will then write out the character counts table obtained in the first part of this algorithm (including the counts of 0). Then I will parse through the input file and write each character's encoded value to the compressed file (using the Huffman table) and repeat this process until I reach the end of the input file. Then I will manually write the pseudo-EOF character to the file. I now have my compressed file.

For the decompression program:

Recreate Huffman Tree

I will read the header of the compressed file and recreate the Huffman tree using the same algorithm as above.

Writing to output (uncompressed file)

Once I have the optimal Huffman tree (which will be exactly the same tree as before for each unique

input), I am now ready to start decompressing the file. I will parse through the compressed file and use the Huffman tree to print out the appropriate character to the ouput every time I reach a leaf node. I repeat this process until I reach the psuedo-EOF character. I now have my decompressed file.