Jiacheng Yuan
ECE368
Project 2
10/20/2015

Milestone 1

Our goal in this project is to design a simple version of a file compression and decompression utility by using queues and trees. We are using huffman coding algorithmic technique to generalize compression and decompression.

We are using binary tree with ASCII coding to transform characters into binary value in order to perform the compression. All the characters are stored at the leaves of a tree when using binary tree for coding. On a binary tree, the left-edge is numbered 0 and the right-edge is numbered 1. The ASCII code for any character/leaf is obtained by following the root-to-leaf path and concatenating the 0's and 1's. The specific structure of the tree determines the coding of leaf node and different tree will generate different coding.

The Huffman coding we are going to use was invented by David A. Huffman in 1952 when he was a Ph.D. student at MIT. To construct the tree using Huffman's algorithm we assume each character has a weight associated with the number of times of the character's occurrence in file. Begin with a forest of trees which all trees just have one node, with the weight of tree equal to weight of the character in the node. Create a new tree whose root has a weight equal to the sum of two trees until there is only one tree. This single tree is an optimal encoding tree. The two parts in implementing Huffman coding are compression program and decompression program. To compress a file, we need build a table of per-character encodings first. To build a table of optimal per-character bit sequences, we need count the number of times every character occurs and create an initial forest of one-node trees with the weight. Then form a single tree by using the greedy Huffman algorithm. Then follow every root-to-leaf path of the tree to create the table of bit sequence encodings. With the table, we can read the file and process the characters one at a time. There must include a header information which initially stored in the compressed file that will be used by the decompression program. We must store the tree that is used to compress the original file for the decompression program because it needs exact same tree in order to decode the data. We could either store the character counts or store the tree at the beginning of the file.

On many systems, all file sizes are multiples of 8 or 16 bits, so that it is not possible to have a 122-bit file. So our decompressing program must have some mechanism to account for these extra bits since these bits do not represent compressed information. We can use a pseudo-EOF character and write a loop that stops when the pseudo-EOF character is read in. When a compressed file is written, the last bits written should be the bits that correspond to the pseudo-EOF character.

Our task is to write two programs which perform compression and decompression. The compression program should accept one argument of input file. The program compress the input

file using Huffman coding and write output to a file with same name and append ".huff" to it. The "unhuff.c" program which performs decompression and it will also accept one argument which will be the name of input file. The program decompressed the file and write output to a file with same name and append ".unhuff" to it.