Algorithms

For the huff part program, I first read the whole input file once in order to count the symbol and frequency of them. Then I use a user define function to build the tree follow the rule that high frequency with low height. After these I just sample print the tree and compressed information out.

For the unhuff part, I read the tree information first and build the tree. Then I read the other information to decompress the file.

The way I compress the file

For the tree I want to save in the output file, I use 1 and 0 to show the way it passed. 0 means the tree move to the left node and 1 means it move to the right side. When it reaches the leaf, if will print the symbol in that node. I use a space to distinguish the way and actual symbol since I find in some file, there are 1 and 0 need me to compress. For example, a tree has two leaf, a and b, will print out 1a0b. To print out all the information in the tree, I use the recursion to build the function.

After I print the tree, I begin to read the symbol in the file. Every time I read one character and turn them in to the 1 and 0 by using the tree I build before. When there are eight or more 1 and 0 symbol. I will take first 8 out and turn them in to decimal place. Then use ASCII to print the decimal in to the output file. I will repeat this until the input file reach the end. If the last 1 and 0 symbol at the end is not enough, I will add some 0 to make it to eight.

The way I decompress the file

For the tree I want to rebuild, I just read the 1 and 0. If it is 1, move to the node right of the tree. If it is 0, move to the left. When read a space, read next one and save it to the node I am in. then return to the header of the tree and start to get new. If there is a space after the character we saved, it means the tree reach the end.

For the decompress information, I just read each number once and break them in to eight bits 1 and 0 number and then save them in to the array. After I read all the information, I go through the array from beginning and follow the I and 0 to read the tree. If it reaches the leaf and print the character, tree will go back to the header and do another search.

Performance
Time: I didn't actual timing my program. But both of my huff and unhuff program run fast for the input you provided. Huff part works a little bit longer than the unhuff since I use a lot of loop and recursion in it.
Compress: The compress is good for the file that larger than 1 KB. Since the way I save the tree will use many bytes. But since the tree just have at most two hundred or little more of symbol, it won't influence the file larger than 1 KB. For the file you provide. I can compress them around to 2/3 of original size. The 3MB file to 2.2MB, 6 to 4.4 and 9 to 6.6.

Problem: I meet some problem in my code. First, my huff code will segmentation fault at the end of main at the return place. Second, for the given input text1.txt, my decompressed output will always miss one byte by the output file is totally same as the original file. Third, for some input, the output will have one more bytes. This is because the 0 I put into it at the end of compress.