Huffman Coding Project Technical Design

Victor W. Frye

Davenport University

Huffman Coding Project Technical Design

# Overview

This program will compress and decompress an input ASCII text file using the Huffman coding algorithm.

# Scope

This program will compress or decompress an input ASCII text file using the Huffman coding algorithm. Execution of the program will be done from the command line and determine whether to compress or decompress based on the input arguments.

The input file for compression will be read as-is and not converted in any way. The input file is assumed to be an ASCII text file and is not translated in any way. The output file for compression will be plain binary text. Additionally, a key file will be generated to decompress the file to its original state.

The input file for decompression will be assumed to be a compressed file and will not be translated in any way. The input file is assumed to be a encrypted ASCII text file with only binary representation of the original data. A key file will be required to decompress the input file. If no key file is found, the input file will be read as corrupted.

# Usage

To use the program, two additional command line arguments will be needed compared to the standard executable calling. The first will either be “compress” or “decompress”, designating what action to perform upon the file. The second argument is the file path for the input file.

Note that the output file and key file are defaulted within the program. The output file will output as “output.txt” and the key file will output as “key.txt”.

## Processing Outline

1. If the number of command line arguments is not two
   1. Display usage error message
   2. Exit
2. If the action route argument is invalid (not compress or decompress)
   1. Display usage error message
   2. Exit
3. If the input file does not exist
   1. Display input error message
   2. Exit
4. Initialize an array of character frequency objects
   1. For each element, create a character frequency object with the character set to the corresponding ASCII value based upon the index
   2. Set the initial character frequency count to zero for each element in the array
5. If action route argument is to compress
   1. Open the input file
      1. Read a character from the input file
      2. While not at the end of the input file
         1. Increment the character frequency count using the ASCII value of the character as the index
         2. Read the next character
   2. Close the input file
   3. Sort the array of character frequency objects using the built-in array sort method
   4. Initialize a linked list of nodes of type character frequency
      1. For each element in the array of character frequency objects
         1. If the frequency of the character frequency object is not zero (0)
            1. Initialize a node with the character frequency object inserted as the root of the tree
            2. Add the newly created node to the end of the linked list
   5. Construct a binary tree of the character frequency objects
      1. While the count of linked list nodes is less than one (1)
         1. If only two nodes remain in the linked list and the first node is larger than the last
            1. Swap the first and last nodes
         2. Remove the first two (2) nodes from the linked list
         3. Construct a new node with a frequency equal to the sum of the frequencies of the two (2) removed nodes
         4. Connect the three (3) nodes, with the newly constructed node being the parent and the two (2) removed nodes being the children (the smallest should be on the left)
         5. If the new node is smaller than all other nodes or no nodes exist in the linked list
            1. Add the new node to the beginning of the linked list
         6. Else find the first node in the linked list that is larger than the new node and add the new node before it
   6. Construct an encoding table
   7. Create compressed file
      1. Open the input file
      2. If the output file exists
         1. Delete the output file
      3. Open the output file
      4. Read a character from the input file
      5. While not at the end of the input file
         1. Find assigned Huffman code for the read character
         2. For each bit in the code
            1. If a full byte has been constructed

Write byte to output file

Reset byte and counter

* + - * 1. If the current bit in the code is a one (1)

Modify the bit corresponding to counter-designated place in the byte to be a one (1)

Increment the counter

* + - * 1. Else increment the counter
      1. Read next character from the input file
    1. Write final byte to the output file
    2. Close the output file
  1. If key file exists
     1. Delete the key file
  2. Open the key file
     1. For each character frequency object in the character frequency array
        1. If the frequency of the object is not zero (0)
           1. Write a string representation of the character frequency object to a new line in the key file
  3. Close the key file

1. Else (assumed to be decompression action route)
   1. If the key file does not exist
      1. Display missing key file error message
      2. Exit
   2. Open the key file
      1. Instantiate counter for total characters
      2. Read the first line from the key file
      3. While the read line is not null
         1. Check read line for matching string representation of a character frequency object
         2. Convert matching string representation to character frequency objects if found
         3. Add the found frequency to the total characters counter
         4. Read the next line
   3. Close the key file
   4. Initialize a linked list of nodes of type character frequency
      1. For each element in the array of character frequency objects
         1. If the frequency of the character frequency object is not zero (0)
            1. Initialize a node with the character frequency object inserted as the root of the tree
            2. Add the newly created node to the end of the linked list
   5. Construct a binary tree of the character frequency objects
      1. While the count of linked list nodes is less than one (1)
         1. If only two nodes remain in the linked list and the first node is larger than the last
            1. Swap the first and last nodes
         2. Remove the first two (2) nodes from the linked list
         3. Construct a new node with a frequency equal to the sum of the frequencies of the two (2) removed nodes
         4. Connect the three (3) nodes, with the newly constructed node being the parent and the two (2) removed nodes being the children (the smallest should be on the left)
         5. If the new node is smaller than all other nodes or no nodes exist in the linked list
            1. Add the new node to the beginning of the linked list
         6. Else find the first node in the linked list that is larger than the new node and add the new node before it
   6. If the output file already exists
      1. Delete the output file
   7. Open the input and output file
      1. Read a character from the input file
      2. Instantiate counter and array to measure out full bytes
      3. While input file data is valid
         1. If a full byte has been measured out
            1. Search the binary tree for leaf nodes corresponding to path designated by bits

If bit is a one, traverse the tree to the left

If bit is a zero, traverse the tree to the right

If a leaf node is found and the total remaining character to print are greater than zero

Write the character corresponding to the leaf node to the output file

Return to the root of the tree

Decrease the remaining amount of characters in file

* + - * 1. Reset the counter and byte measuring array
      1. Else
         1. Add bit to the byte measuring array
         2. Increment counter
         3. Read the next character from the input file
    1. Close the input and output file

1. Exit

# Data

## Input File

The input file will either be a standard ASCII text file or a compressed binary bit text file depending on whether compression or decompression is the current action.

For compression, it will be assumed that the file is an ASCII text file and will be read a single character at a time in sequential order. If a file other than an ASCII text file is used, the results will be undefined.

For decompression, it will be assumed that the file was compressed using this program and a key file already exists. If no key file is found, the program will reject the process. The input file will be read bit by bit looking for ones (1) and zeros (0). No other characters or information will be recognized. If a file other than one compressed by this program is used, the results will be undefined.

## Output File

The output file for compression will be a series of ones (1) and zeros (0) as the output is a binary representation of the original data. The output of the decompression process will be the original file assuming that the key file was not corrupted or tampered with.

## Data Structures

An array containing 256 character frequency objects will be used in both compression and decompression to determine the frequency of the characters found in the original file. The decompression process will rely on a key file to gather the frequencies of the characters in the original file. Regular expression will be used to decipher the key file in this process. Access to the array is handled by perfectly hashing the ASCII value of the character to the index of the array. After all frequencies have been determined, the array is sorted using the built-in sort method and is no longer able to perform flawless O(1) operations.

The sorted array is instantiated into a linked list of all character frequency objects with greater than zero occurrences registered. The linked list is used allow for quick removal of low frequency nodes to create a binary tree. A linked list is used rather than a stack so that composite nodes (nodes with a summed frequency of two smaller nodes) are still able to be re-inserted into the linked list in ascending order. This reinsertion is an O(n) operation that later allows for quick access through the binary tree.

A binary search tree is the primary data structure used to perform the Huffman coding process. The binary search tree branches with smaller nodes on the left and larger nodes on the right. This allows for the Huffman code designation by assigning a one or zero respectively to the traversal.

Footnotes

1Note: For a visible representation of the ASCII table, see Figure 1. Since some ASCII characters are invisible, they are identified as a string in the table, e.g. “NUL”, “SOH”, “BEL”, and “CR”.

Tables and Figures

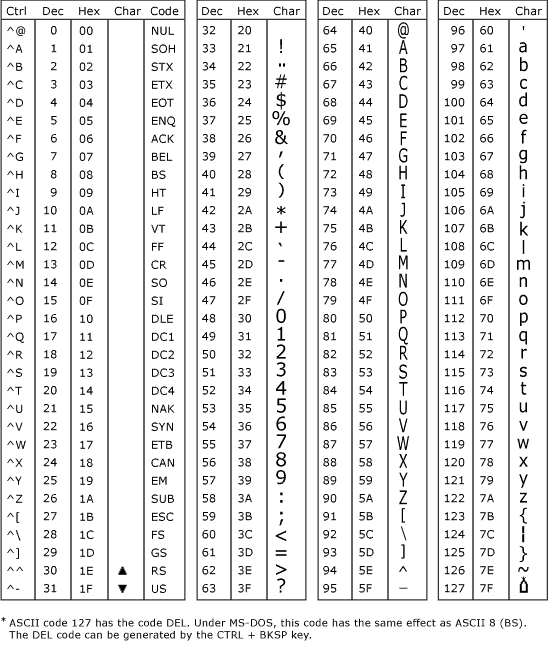
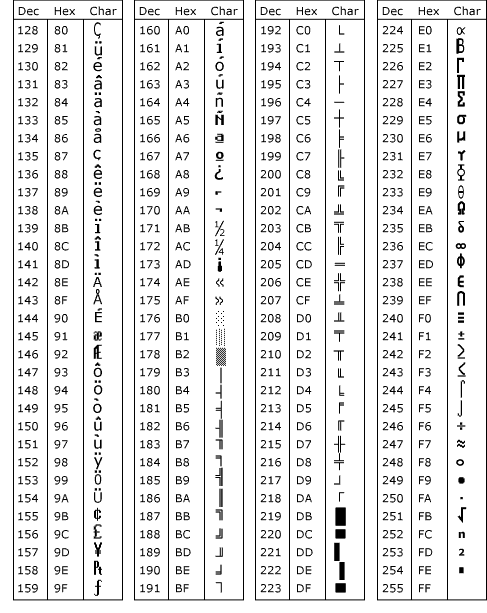


Figure 1. The American Standard Code for Information Interchange (ASCII) table. The above images contain the decimal and hexadecimal values and their corresponding character. The charcters above 128 in decimal are known as the extended character set.