Project

Algorithms in Computational Biology - BIO522

Text file compression and decompression using Huffman Coding

Umang Sharma(MT23239), Mimansha Das(MT23233), Sarvani Gupta(MT23252)

READ ME

Overview

This program implements Huffman coding, a popular method for lossless data compression. The program includes functions to compress a file using Huffman encoding and then decompress the file back to its original state. Huffman coding is efficient for encoding data with varying frequency of characters, achieving better compression ratios than fixed-length encoding schemes.

Requirements:

C++ with MinGW as compiler

Features

Compression: Reads a file, constructs a frequency table for its characters, builds a Huffman tree, and generates the Huffman codes. The compressed data is then written to an output file along with a header containing the Huffman codes.

Decompression: Reads the compressed file, extracts the Huffman codes from the header, and reconstructs the original file from the compressed bitstream.

Components

Data Structures

- Tree: Represents a node in the Huffman tree.
- int frequency: The frequency of the character.
- unsigned char character: The character the node represents.
- Tree *left, *right: Pointers to the left and right child nodes.
- TreeComparator: A comparator class used to order nodes in the priority queue based on their frequencies.

Functions

- buildHuffmanTree:

Constructs the Huffman tree from a frequency table.

Uses a priority queue to combine the least frequent nodes.

- toBinary:

Converts a character to its binary string representation.

- traverseHuffmanTree:

Recursively traverses the Huffman tree to generate the Huffman codes for each character.

- readFileIntoBuffer:

Reads a file into a buffer.

Returns a pointer to the buffer and the size of the buffer.

- writeFileFromBuffer:

Writes data from a buffer to a file.

- convertToVector:

Converts a frequency map to a vector of pairs.

- getHuffmanBitstring:

Generates a bitstring for the entire file using the Huffman codes.

Pads the bitstring to make its length a multiple of 8.

- getBufferFromString:

Converts a bitstring to a buffer of bytes.

- getStringFromBuffer:

Converts a buffer of bytes to a bitstring.

- getDecodedBuffer:

Decodes a bitstring using the Huffman codes to reconstruct the original data.

- writeHeader:

Writes the Huffman codes and padding information to the output file.

- readHeader:

Reads the Huffman codes and padding information from the input file.

- compressFile:

Handles the entire process of reading a file, generating Huffman codes, compressing the file, and writing the compressed data to an output file.

- decompressFile:

Handles the entire process of reading a compressed file, extracting Huffman codes, and reconstructing the original file.

Main Function

The main function provides an interface for compressing and decompressing files via command-line arguments.

Usage of Code:

Compilation

To compile the program, use a C++ compiler such as g++:

bash

```
g++ -o huffman huffman.cpp
```

Execution

To run the program, you can specify the input file, the compressed output file, and the decompressed output file:

bash

```
./huffman <input_file> <compressed_file> <decompressed_file>
```

If no arguments are provided, the program defaults to using "test.txt" for input, "test1.txt" for the compressed output, and "decoded.txt" for the decompressed output.

Example

bash

```
./huffman example.txt compressed.huff decompressed.txt
```

This will:

Compress example.txt to compressed.huff.

Decompress compressed.huff to decompressed.txt.

Detailed Description

Huffman Tree Construction

The buildHuffmanTree function creates a Huffman tree from a frequency table. It uses a priority queue to ensure that the nodes with the lowest frequencies are combined first. Each node in the tree represents either a character from the input file or a combination of characters.

Code Generation

The traverseHuffmanTree function recursively traverses the Huffman tree to generate a unique binary code for each character. These codes are stored in a map, where each character is mapped to its corresponding Huffman code.

Compression

The compressFile function orchestrates the compression process:

Reads the input file into a buffer.

Builds the frequency table.

Constructs the Huffman tree and generates the codes.

Creates a bitstring for the entire file using the Huffman codes.

Pads the bitstring to ensure its length is a multiple of 8.

Writes the Huffman codes and the bitstring to the compressed file.

Decompression

The decompressFile function handles the decompression:

Reads the compressed file into a buffer.

Extracts the Huffman codes and padding information from the header.

Converts the remaining data in the buffer back into the original bitstring.

Uses the Huffman codes to decode the bitstring and reconstruct the original file.

Header Format

The header of the compressed file contains:

The number of padding bits.

The number of unique characters.

Each character is followed by the length of its Huffman code and the code itself.

This information allows the decompression function to reconstruct the Huffman tree and decode the bitstream accurately.

Conclusion

This Huffman compression and decompression program is a practical implementation of one of the most efficient lossless compression algorithms. By leveraging the properties of variable-length codes, it effectively reduces the size of files containing characters with varying frequencies. This program serves as a robust tool for anyone needing to compress and decompress files using Huffman coding.