A Project Report

On

Huffman Coding Algorithm

Submitted in partial fulfillment of the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY

GALGOTIAS UNIVERSITY

(Established under Galgotias University Uttar Pradesh Act No. 14 of 2011)

Session 2023-24

By

Ayush Agarwal {21SCSE1010588}

Anshul Agarwal {21SCSE1010901}

Aaditya Srivastava {21SCSE1011522}

Under the guidance of

Dr. Vimal Kumar

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GALGOTIAS UNIVERSITY, GREATER NOIDA

INDIA

Jan, 2024

INTRODUCTION

The Huffman Coding Compression project aims to implement a simple file compression and decompression system using the Huffman coding algorithm. The system allows users to upload a text file, compress it using Huffman coding, and then decompress it back to its original form.

Project Objectives

- Implement Huffman coding for file compression.
- Develop a web-based user interface for file upload and interaction.
- Display compressed and decompressed file paths.
- Show original, compressed, and decompressed file sizes.

Technologies Used

Programming Language: Python

Web Framework: Flask

System Architecture

The system follows a client-server architecture. The server is implemented using Flask, handling file upload, compression, and decompression operations. The client is a simple HTML interface allowing users to interact with the system.

The Huffman coding algorithm efficiently represents symbols in the input data with variable-length codes, assigning shorter codes to more frequent symbols. This results in a compressed representation of the original data, reducing file size without loss of information. The provided Python code encapsulates these Huffman coding steps in the `HuffmanCoding` class, making it reusable for file compression and decompression.

Implementation

Huffman Coding Implementation

The `encode.py` file contains the implementation of the Huffman coding algorithm. It includes classes for creating a Huffman tree, encoding text, and compressing/decompressing files.

Web Interface Implementation

The `useEncode.py` file utilizes Flask to create a web interface. The `/` route serves the HTML form for file upload, and the `/upload` route handles file upload, compression, and decompression operations.

Usage:

Run the Flask application: `python useEncode.py`

1. Access the application in a web browser: `http://127.0.0.1:5000/`
2. Choose a text file, click “upload file” and view the results.

Results

Compressed file path: `<path_to_compressed_file>`

- Decompressed file path: `<path_to_decompressed_file>`
- Original file size: `<original_size>` bytes
- Compressed file size: `<compressed_size>` bytes
- Decompressed file size: `<decompressed_size>` bytes

Huffman Coding Algorithm Overview:

Frequency Calculation:

- The algorithm starts by calculating the frequency of each symbol (in this case, characters) in the input text. A frequency dictionary is created to store the count of each symbol.

2. **Priority Queue (Min Heap) Creation:**

- A priority queue, often implemented as a min-heap, is used to store nodes representing symbols along with their frequencies. The priority queue ensures that nodes with lower frequencies are given higher priority.

3. **Building the Huffman Tree:**

- Nodes are continuously removed from the priority queue. Two nodes with the lowest frequencies are combined to create a new node, and this new node is inserted back into the priority queue. This process continues until only one node (the root of the Huffman tree) remains in the priority queue.

4. **Generating Huffman Codes:**

- Huffman codes are generated by traversing the Huffman tree. During traversal, a binary code is assigned to each symbol. Typically, moving to the left child represents appending "0" to the code, and moving to the right child represents appending "1."

5. **Compression:**

- The original text is replaced with its corresponding Huffman codes. This results in a compressed representation where more frequent symbols have shorter codes.

6. **Decompression:**

- During decompression, the compressed bitstream is traversed using the Huffman tree. As the tree is traversed, the original symbols are reconstructed.

Output


127.0.0.1:5000
File Compressor ~ Huffman Coding
input File:
Choose File
No file chosen
Upload File
Original file size: 612 bytes
Compressed file path: uploads/Rule book.bin (345 bytes)
Decompressed file path: uploads/Rule book_decompressed.txt (611 bytes)