**Project Report – CSET202**

**PROJECT TITLE :** TEXT FILE COMPRESSOR

* TEAM MEMBERS: ADITYA RAJ, SANJANA NAYAK
* ENROLLMENT NO: E22CSEU0649, E22CSEU0644
* BATCH: 22

INTRODUCTION

Our project aims to develop a file compressor using C++ data structures. It uses various algorithm ex:

* Huffman Coding
* Run-Length Encoding (RLE)
* Lempel-Ziv-Welch (LZW)
* Burrows-Wheeler Transform (BWT), etc.

**File compression** is a process that reduces the size of one or more files to save storage space and make data transfer more efficient. It works by encoding the data in a way that reduces redundancy and eliminates unnecessary information, resulting in a smaller file size while preserving the essential data and allowing for easy decompression to its original form.

It helps in Storage Efficiency, Faster data transfer. They are used in various industries and scenarios such as: data backup, web content, email attachments, software distribution.

*In this project we will talk about the compression ratios which measure the compressed algorithm. For example, a 2:1 compression ratio means the compressed file is half the size of the original.*

RUN LENGTH ENCODING (RLE)

HISTROY

The concept of RLE can be traced back to the early days of telegraphy and teletype machines. Operators used RLE to transmit messages more efficiently by encoding repeated characters.

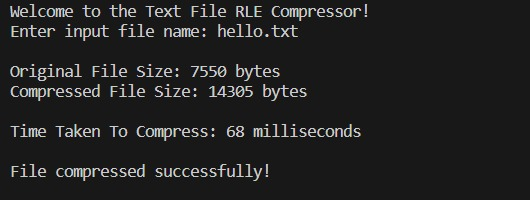
INTRODUCTION

Run-Length Encoding (RLE) is a simple and widely applied data compression technique, and it has seen contributions and applications from Early Telegraph Operators, Computer Scientists and Engineers, and Graphics and Imaging Experts, Biologists and Bioinformaticians, etc.

GAPS

* Ineffective for Random Data in which RLE is not suitable for random or highly varied data. In such cases, it may result in larger encoded files or no compression at all, as there are few consecutive runs of identical elements to compress.
* Limited Compression for Complex Data: RLE may not effectively compress data with complex patterns or variations, such as photographs, audio, or video. These types of data typically do not have long runs of identical elements, making RLE inefficient.

ANALYSIS OF RESULT



REFERENCE

1. W. Burange, H. D. Misalkar, “Review of Internet of Things in development of smart cities with data management & privacy,” in 2015 International Conference on Advances in Computer Engineering and Applications (ICACEA), (2015) pp. 189 - 195.
2. S. Abraham, J. Beard, R. Manijacob, “Remote Environmental Monitoring Using Internet of Things (IoT).” 2017 IEEE Global Humanitarian Technology Conference (GHTC), (2017).

**------------------------------------------------------------------------------------------------------------------**

HUFFMAN CODING  
HISTROY

The history of Huffman coding is closely tied to the work of David A. Huffman, who developed the algorithm as part of his Ph.D. thesis at MIT.

INTRODUCTION

Huffman coding, also known as Huffman compression, is a widely used and efficient data compression technique that assigns variable-length codes to input characters or symbols based on their frequencies in the data. Frequency Count: Calculate the frequency of each character or symbol in the input data. Building a Huffman Tree-Binary tree or Huffman Tree is constructed by iteratively merging the two nodes with the lowest frequencies into a parent node. This process continues until all nodes are merged into a single root node.

GAPS

* Lossless Compression Only: Huffman coding is a lossless compression method, meaning it preserves all the original data. While this is essential for certain applications, it may not be suitable for scenarios where lossy compression can be applied.

ANALYSIS OF RESULT

A computer screen shot of a black screen

Description automatically generated

REFERENCE

1. Web Page Compression using Huffman Coding Technique Manjeet Gupta (Assistant professor) Department of CSE JMIT Radaur , Brijesh Kumar (associate professor) Department of IT Lingyaya’s university.
2. Fundamental Data Compression , Author(s): Ida Mengyi P

**------------------------------------------------------------------------------------------------------------------**

LEMPEL-ZIV-WELCH (LZW)

HISTROY

It was developed by Abraham Lempel and Jacob Ziv and further refined by Terry Welch. The LZW algorithm has a rich history and has made significant contributions to data compression techniques. The LZ algorithm was a breakthrough because it allowed for dictionary-based compression, which was more adaptable to various types of data. The LZW algorithm was further extended and refined by Terry Welch in 1984.

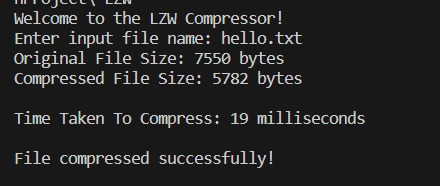
INTRODUCTION

Lempel-Ziv-Welch (LZW) algorithm is a widely used data compression algorithm that forms the basis for various popular compression formats, such as GIF (Graphics Interchange Format) and the early versions of the UNIX "compress" command.

GAPS

* Limited Compression Efficiency for Some Data Types: LZW may not be the most efficient compression method for certain types of data, especially when compared to more modern compression algorithms. It works well for text data and certain patterns, but it might not provide the highest compression ratios for complex data, such as multimedia files.
* Lossless Compression Only: LZW is a lossless compression method, which means it preserves all the original data. It's not suitable for applications that can benefit from lossy compression techniques to achieve higher compression ratios (e.g., multimedia data).

ANALYSIS OF RESULT



REFERENCE

1. S. E. Collier, “The Emerging Enernet: Convergence of the Smart Grid with the Internet of Things,” in 2015 IEEE Rural Electric Power Conference (REPC), (2015) pp. 65-68.
2. [2] A. W. Burange, H. D. Misalkar, “Review of Internet of Things in development of smart cities with data management & privacy,” in 2015 International Conference on Advances in Computer Engineering and Applications (ICACEA), (2015) pp. 189 - 195.
3. S. Abraham, J. Beard, R. Manijacob, “Remote Environmental Monitoring Using Internet of Things (IoT).” 2017 IEEE Global Humanitarian Technology Conference (GHTC), (2017).

**------------------------------------------------------------------------------------------------------------------**

BURROWS-WHEELER TRANSFORM (BWT)

HISTROY

It is named after its inventors, Michael Burrows and David Wheeler, in 1994 while working at Digital Equipment Corporation's Systems Research Center. Their objective was to create a new compression technique that could potentially overcome some of the limitations of existing algorithms like LZW.

INTRODUCTION

The Burrows-Wheeler Transform is a reversible permutation of a block of data. It rearranges the characters in a way that exposes repetitions and similarities, making it an ideal preprocessing step for compression algorithms. The BWT is widely used in various data compression tools and formats, such as the Burrows-Wheeler Block Sort Text Compression (BZIP2) algorithm.

GAPS

* Lossless Compression Only: BWT is a lossless compression technique, which means it preserves all the original data. While this is crucial for certain applications, it's not suitable for scenarios where lossy compression can be applied to achieve higher compression ratios, particularly for multimedia data.
* Encoding Overhead: The BWT itself doesn't necessarily reduce the size of the data; it rearranges it to make subsequent compression more effective. However, the rearrangement can introduce some encoding overhead, which may affect compression efficiency.

ANALYSIS OF RESULT

A computer screen shot of a computer

Description automatically generated

REFERENCE

1. B. Balkenhol, S. Kurtz, and Y. M. Shtarkov. Modiﬁcation of the Burrows and Wheeler data compression algorithm. In DCC: Data Compression Conference. IEEE Computer Society TCC, 1999.
2. J. Bentley and R. Sedgewick. Fast algorithms for sorting and searching strings. In Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, pages 360–369, New Orleans, Louisiana, 1997

**------------------------------------------------------------------------------------------------------------------**

LEMPEL-ZIV 1977 (LZ77)

HISTROY

In the 1970s, researchers and engineers were actively exploring techniques to compress data efficiently, primarily to save storage space and reduce data transfer times. Abraham Lempel and Jacob Ziv, both computer scientists, collaborated to develop a series of data compression algorithms known as LZ algorithms.

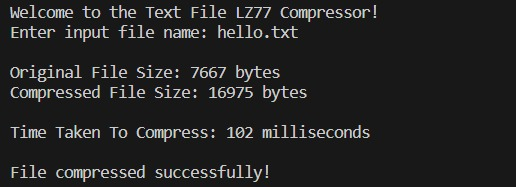
INTRODUCTION

The LZ77 algorithm, introduced by Lempel and Ziv in 1977, is a dictionary-based compression method. It works by replacing repeated sequences of characters in the input data with references to previous occurrences, resulting in a more compact representation.

GAPS

* The reason the compressed file size is coming out bigger when using LZ77 compression for this dataset is likely due to the characteristics of the data and the compression algorithm you are using.
* LZ77 may not be the most effective compression algorithm for this type of structured data, especially if the data doesn't contain many repeating patterns or long sequences that LZ77 can efficiently encode.
* LZ77 works best when there are repeated patterns or sequences of characters in the data. If your dataset consists of numerical values and doesn't have many repeated patterns, LZ77 may not provide significant compression, and it can even lead to a larger compressed file size due to the overhead introduced by the encoding of the positions, lengths, and literals.

ANALYSIS OF RESULT



REFERENCE

1. S. E. Collier, “The Emerging Enernet: Convergence of the Smart Grid with the Internet of Things,” in 2015 IEEE Rural Electric Power Conference (REPC), (2015) pp. 65-68.
2. W. Burange, H. D. Misalkar, “Review of Internet of Things in development of smart cities with data management & privacy,” in 2015 International Conference on Advances in Computer Engineering and Applications (ICACEA), (2015) pp. 189 - 195.

**------------------------------------------------------------------------------------------------------------------**

ARITHEMETIC CODING

HISTROY

The idea of arithmetic coding can be traced back to pioneering work by Claude Shannon in the 1940s. In the 1970s, a number of researchers, including Robert W. Floyd, Peter Elias, and G. N. N. Martin, independently developed arithmetic coding as a practical method for data compression.

INTRODUCTION

Arithmetic coding is a sophisticated and highly efficient data compression technique that encodes data as a single real number in the unit interval [0, 1].

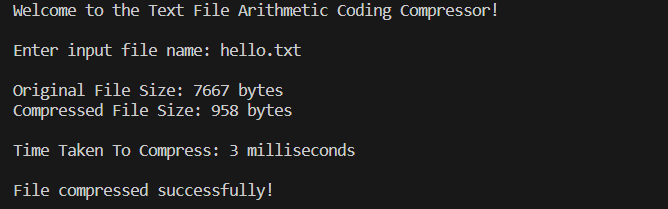
CONTEXT

* Improved Efficiency: Researchers continue to explore ways to improve the efficiency of arithmetic coding, both in terms of compression ratios and speed. This includes refining probability modeling techniques, exploring more advanced data modeling, and optimizing the arithmetic coding process.
* Context-Based Compression: Combining arithmetic coding with context modeling has gained attention. Context modeling considers the context of the data being encoded, enabling more accurate probability modeling and, therefore, better compression ratios.

GAPS

* Parallelization: Efficiently parallelizing arithmetic coding for multi-core processors and hardware accelerators is a challenge. Achieving true parallelism in the encoding and decoding processes is essential for improving performance in modern computing environments.
* Lossy Compression: Expanding the use of arithmetic coding in lossy compression techniques, while maintaining an appropriate balance between compression efficiency and data quality, is an ongoing challenge. This is especially relevant for multimedia compression.

ANALYSIS OF RESULT



REFERENCE

1. S. E. Collier, “The Emerging Enernet: Convergence of the Smart Grid with the Internet of Things,” in 2015 IEEE Rural Electric Power Conference (REPC), (2015) pp. 65-68.
2. W. Burange, H. D. Misalkar, “Review of Internet of Things in development of smart cities with data management & privacy,” in 2015 International Conference on Advances in Computer Engineering and Applications (ICACEA), (2015) pp. 189 - 195.

**------------------------------------------------------------------------------------------------------------------**

BZIP-2

HISTROY

Bzip2 was developed by Julian Seward, a British computer scientist, as an improvement over his previous work with the "bzip" compression utility.

INTRODUCTION

Bzip2, short for "Burrows-Wheeler Block Sorting Text Compressor, Version 2," is a widely used data compression algorithm and associated file format that uses the Burrows-Wheeler Transform (BWT) and Run-Length Encoding (RLE) in its compression process. It is known for its high compression efficiency, making it particularly suitable for compressing text and data files.

CONTEXT

* Unix and Linux Systems: Bzip2 remains a commonly used compression tool in Unix and Linux environments. It is often employed for compressing and decompressing files and directories, especially when efficient compression of text and data files is required.
* Software Distribution: Bzip2 is frequently used in software distribution for compressing source code, software packages, and archives. Many open-source projects and software repositories continue to offer Bzip2-compressed files for download.

GAPS

* Compression Efficiency: Bzip2, while effective, may not always provide the highest compression ratios compared to some newer algorithms like zstd or xz.
* Decompression Speed: Decompression speed in Bzip2 may be slower compared to other compression formats. Optimizing the decompression process to make it faster, especially for large archives, is a potential improvement.

REFERENCE

1. S. E. Collier, “The Emerging Enernet: Convergence of the Smart Grid with the Internet of Things,” in 2015 IEEE Rural Electric Power Conference (REPC), (2015) pp. 65-68.
2. W. Burange, H. D. Misalkar, “Review of Internet of Things in development of smart cities with data management & privacy,” in 2015 International Conference on Advances in Computer Engineering and Applications (ICACEA), (2015) pp. 189 - 195.

**------------------------------------------------------------------------------------------------------------------**

PREDICTION BY PARTIAL MATCHING ( PPM )

HISTROY

The concept of context modeling for data compression can be traced back to the work of Jorma Rissanen and Pekka Langdon in the late 1970s. They introduced the idea of adaptive probabilistic models for encoding symbols in data streams. The first implementations of PPM were PPM-1 and PPM-2, developed by Jorma Rissanen in 1981. PPM-3, developed by John Cleary and Witten in 1984.

INTRODUCTION

Prediction by Partial Matching (PPM) is a family of lossless data compression algorithms. These algorithms use a probabilistic approach to predict and encode the next symbol in a data stream based on the context provided by the previous symbols. PPM is known for its high compression efficiency.

CONTEXT

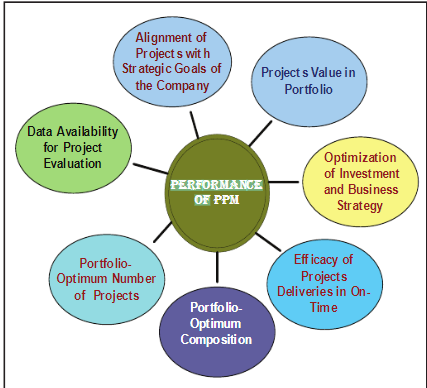
* Text Compression: PPM algorithms, particularly variants like PPM-Z and PPM-D, were widely used for compressing text data.
* Research and Development: PPM algorithms continued to be a subject of research and development.

GAPS

High Memory Requirements: PPM algorithms typically require substantial memory to maintain context models, especially with long contexts.

To address this gap :

* Context Modeling Optimization: Researchers can explore more memory-efficient ways to represent and update context models. Techniques like hash tables, data structures, and adaptive context length can be investigated to reduce memory usage.
* Standardization: PPM algorithms lack standardization compared to well-established compression formats like ZIP and gzip.
* Standardization Efforts: Encourage the development of standardized PPM-based compression formats to facilitate interoperability and broad adoption.



Concept elements of PPM

REFERENCE

1. Cooper, R.G., Edgett, S.J., Kleinschmidt E.J; Best practices for managing R&D portfolios", Research Technology Management, Vol. 41, No. 4, July-Aug. 1998, pp. 20-33.
2. Bigelow, D.,"PM Solutions. Want to ensure quality? Think portfolio management", PM Network, Vol. 17, No. 1, pp.121-122, 2003

**------------------------------------------------------------------------------------------------------------------**

DELTA ENCODING

HISTROY

The idea of encoding data as differences or changes between versions or data items has been around since the early days of computing. In the 1970s and 1980s, the development of version control systems such as RCS (Revision Control System) and SCCS (Source Code Control System) played a significant role in the advancement of delta encoding.

INTRODUCTION

Delta encoding is a data compression technique that stores or transmits the difference (delta) between consecutive data rather than the complete data. It is commonly used in scenarios where consecutive data points have similarities or small variations

CONTEXT

* Data Compression: Delta encoding can be employed in data compression techniques, particularly for data with a significant amount of redundancy or incremental changes. It can be part of a compression strategy to represent differences between data items.
* Version Control Systems: Delta encoding is a fundamental component of version control systems, such as Git, Subversion (SVN), and Mercurial. These systems use delta encoding to store and transmit the differences (or "deltas") between versions of source code files.

GAPS

* High Memory and Processing Requirements: Optimized Data Structures: Researchers and developers can explore more memory-efficient data structures and algorithms for implementing delta encoding, particularly when dealing with large datasets. This can help reduce memory and processing requirements.
* Lossless Compression Only: Lossy Delta Encoding: Consider the development of lossy delta encoding techniques that can be used in scenarios where some loss of data quality is acceptable. This approach can lead to higher compression ratios in some applications.

ANALYSIS OF RESULT

A screen shot of a computer

Description automatically generated

REFERENCE

1. Martin F. Arlitt and Carey L. Williamson. Web Server Workload Characterization: The Search for Invariants (Ex-tended Version). DISCUS Working Paper 96-3, Dept. of Computer Science, University of Saskatchewan, March 1996.

ftp://ftp.cs.usask.ca/pub/discus/paper.96-3.ps.Z.

1. G. Banga, F. Douglis, and M. Rabinovich. Optimistic Deltas for WWW Latency Reduction. In Proc. 1997 USENIX Technical Conf., pp. 289-303. Anaheim, CA, January 1997

**------------------------------------------------------------------------------------------------------------------**

**Comparison Of The Text Compression Algorithms We Have Worked On :**

*We are using a text file with data in tabular form having health info about peoples of a city (****info includes : Age, BMI, Glucose , etc.)***

|  |  |  |  |
| --- | --- | --- | --- |
| Run Length Encoding  (RLE) | 68 | 14305 | -89 |
| Huffman Coding | 20 | 3609 | 52.1 |
| Lempel-Ziv-Welch  (LZW) | 19 | 5782 | 23.4 |
| Burrows-Wheeler Transform  (BWT) | 140 | 5570 | 26.2 |
| Lempel-Ziv 1977  (LZ77) | 102 | 16975 | -125 |
| Arithmetic Coding | 3 | 958 | 87.52 |
| BZIP-2 |  |  |  |
| Predictions By Partial Matching  (PPM) |  |  |  |
| Delta Encoding | 58 | 13648 | -94.41 |

***Original File Size : 7550 bytes***

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm Used For Text Compression** | **Time Taken To Compress**  **(milliseconds)** | **Compressed File Size**  **(bytes)** | **File Size Reduced By**  **(%)** |