**Jaypee Institute of Information Technology, Noida**

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING AND INFORMATION TECHNOLOGY

  
  
  
**Project Title:** ZipWizard

|  |  |
| --- | --- |
| **Enrollment No.** | **Name** |
| 992401030360 | Pragnya Khandelwal |
| 992401030361 | Pratishtha Singh |
|  |  |
|  |  |

Course Name: Software Development Fundamental

Course Code: 24B15CS111

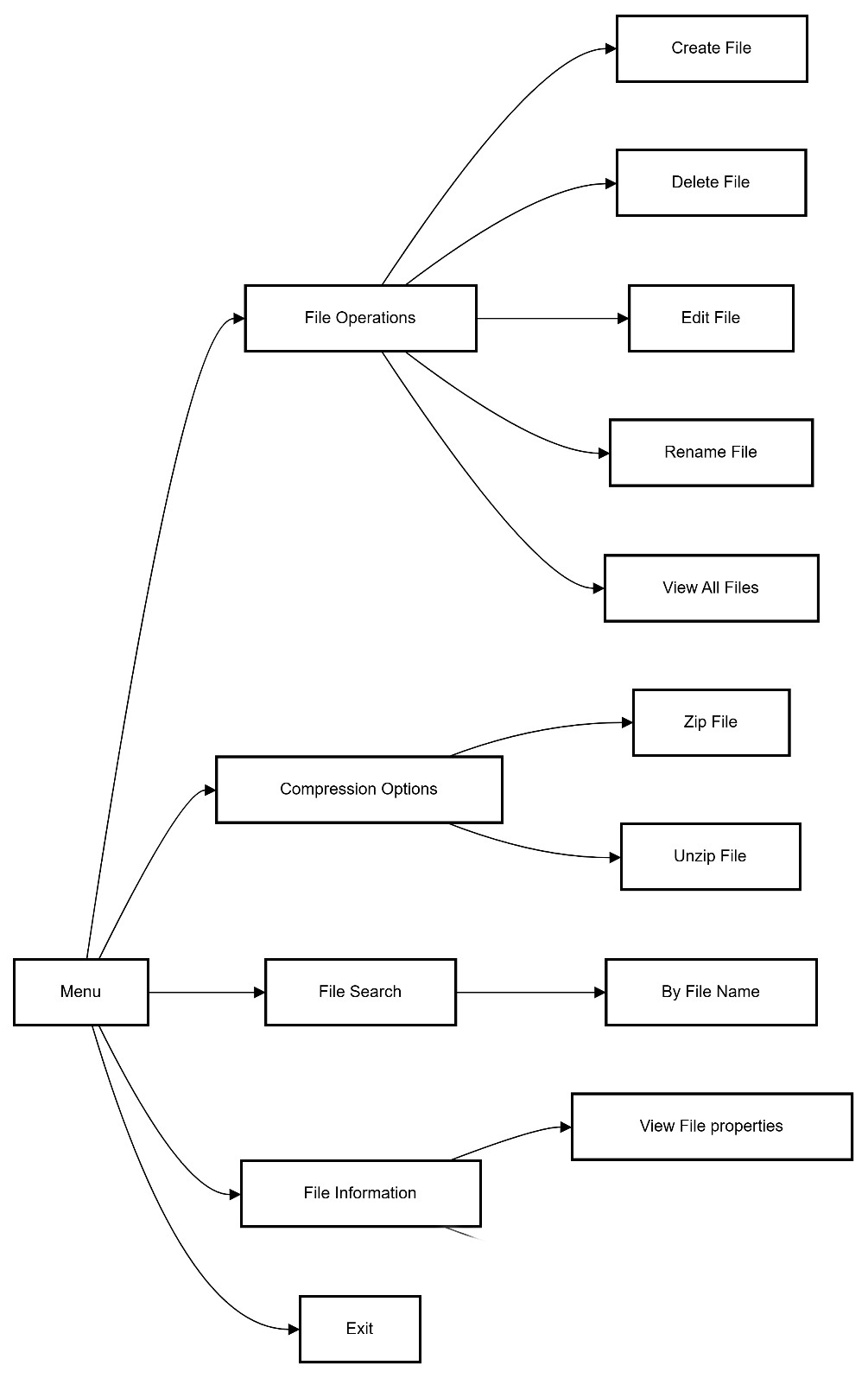
Program: B. Tech. CSE\IT\ECE

1st Year 1st Sem

**2024 - 2025**

**ZIP WIZARD - file handling and compression system**

**Flow – Chart**



**INTRODUCTION**

**Aim:**To prepare a File Handling and Compression System.

**About the project:**

This project, **ZipWizard**, provides a comprehensive solution for handling file operations and implementing basic file compression techniques. The system allows users to create, edit, read, rename, delete, list, search and manage files through an intuitive command-line interface. These operations will be done with the help of different file handling functions in C. In addition to these essential file operations, the system integrates file compression and decompression using two prominent algorithms: **Deflate and Inflate** which is a combination of **Huffman Coding** and **LZ77 compression**.

**Features of the Project:**

**File Operations**:

* **Create**:
  + Users can create new text files by specifying a filename and entering initial content. The system ensures that the filename is valid and does not already exist to avoid overwriting.
* **Read**:
  + Users can read and display the contents of existing text files. The system handles reading line by line or in chunks to accommodate large files and provides options for viewing file metadata, such as size and last modified date.
* **Edit**:
  + Users can modify the content of existing text files, appending to the current content. A simple text editor interface allows users to navigate and make changes easily.
* **Rename**:
  + Users can rename files, with checks to prevent name collisions and to ensure valid naming conventions. Confirmation prompts help avoid accidental changes.
* **Delete**:
  + Users can delete files, with a confirmation step to prevent accidental deletions. Feedback is provided on the success or failure of the operation.

**Searching Files**:

* **Search Functionality**:
  + Users can search for specific text files in the directory by name or by using keyword searches within file contents.
  + The search results display matching files, including relevant details, helping users quickly locate desired files.

**File Properties**:

* **Display File Properties**:
  + The system provides functionality to display detailed properties of a selected file. This includes:
    - **File Name**: The name of the file.
    - **File Size**: The size of the file in bytes.
    - **Creation Date**: The date and time when the file was created.
    - **Last Modified Date**: The date and time when the file was last modified

**Listing Files**:

* + Users can view file attributes, such as date created, and last modified date, making it easier to manage and organize files.

**Compression and Decompression**:

* + The system implements LZ77 coding for efficient compression.
* **Inflate Algorithm**:
  + Decompression functionality restores compressed files to their original state, reconstructing text using LZ77 references.

**User Interface**:

* **Command-Line Interface**:
  + An intuitive, menu-driven command-line interface guides users through operations, with context-sensitive help available for each function.

**File Type Support**:

* **Text Files**:
  + The project specifically handles .txt files, ensuring optimal performance for text-based data.

**Error Handling**:

* Comprehensive error checking is implemented for all operations, providing meaningful feedback to users and enhancing robustness.

**Documentation and Help**:

* Detailed documentation outlines how to use each feature, with built-in help commands accessible directly from the interface.

**Performance Optimization**:

* Performance benchmarks for compression and decompression speeds may be included, along with customizable performance settings.

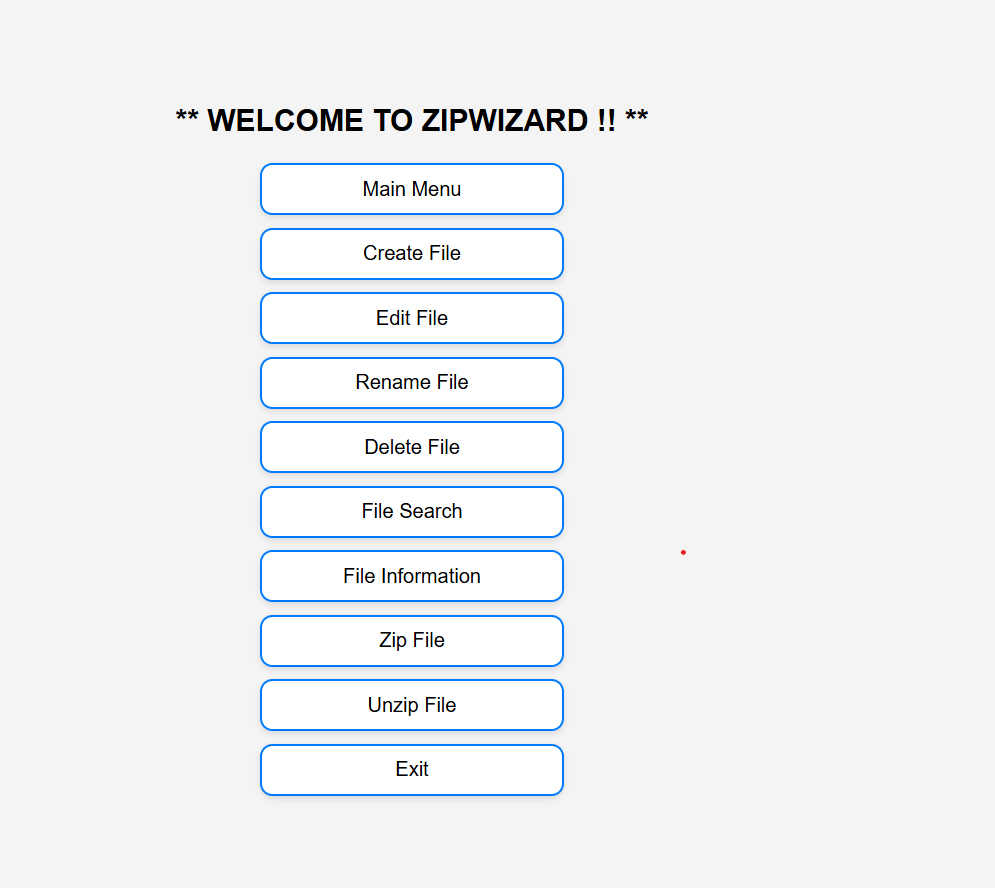
**C - Programming Concepts to be used are:**

**File Handling**: create and manipulate file streams, allowing for operations like opening, reading, writing, and closing files

* **Arrays:** Arrays are used to store character data, frequency counts for Huffman coding, and to maintain lists of files for management and searching.
* **Pointers:** providing flexibility in memory management, enabling efficient data structure manipulation, and allowing direct modification of variables through function parameters.
* **Structures**: Custom structures may be defined to manage file metadata and compression data.
* **Dynamic memory allocation**: Used for allocating and deallocating memory for file contents, Huffman trees, and other structures at runtime, allowing flexibility in handling variable-sized data.
* **User-defined Functions**: The project is organized into functions that handle specific tasks, promoting code reusability and clarity.
* **String Manipulation:** Functions from <string.h> are used for handling strings, including filenames and file contents.
* **Standard Libraries:** Libraries like <stdio.h>, <stdlib.h>, <string.h>, etc. are used for common functions and types needed for file handling, memory management, string manipulation, and time-related operations
* **Error Handling**: The project checks return values of file operations to handle errors efficiently.

**DESIGN OF THE PROJECT**

The design of the project includes the main menu with sub-options that allow users to navigate and perform tasks. Below is the flowchart for better visualization:



**1. System Overview**

**ZIPWIZARD** is a file management tool that allows users to perform the following operations:

* **File Operations**: Create, Edit, Rename, Delete, and View Files.
* **Compression Operations**: Zip and Unzip files.
* **File Information**: Display file metadata, including file name, creation date, and last modified date.
* **File Search**: Search for files by name or content.

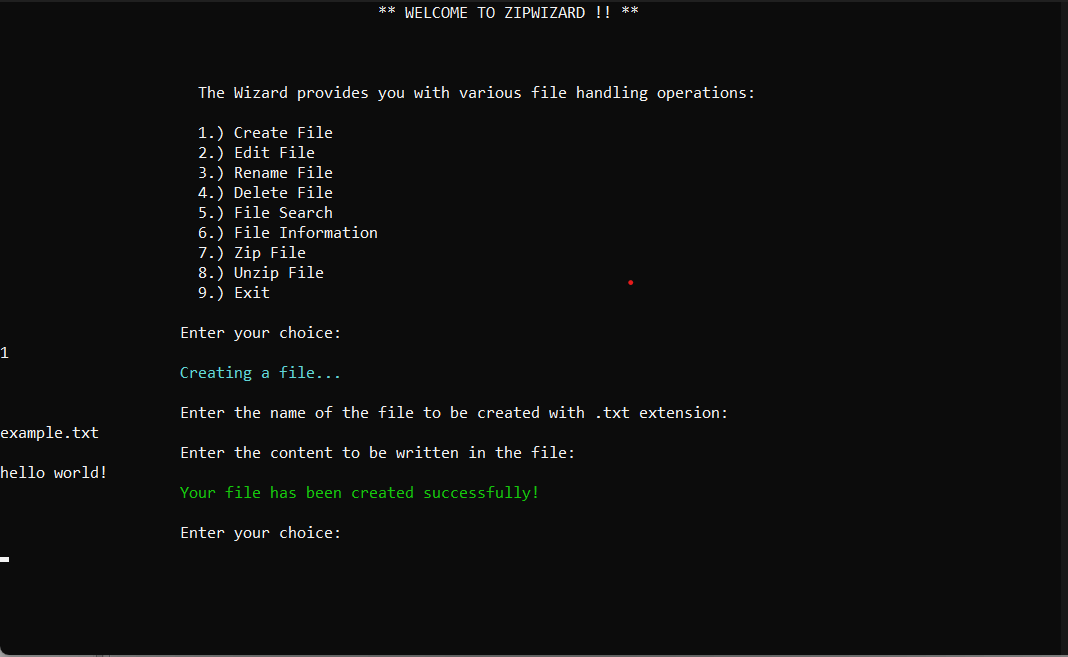
The system will be implemented in **C**, and it will handle text files (.txt files). It will include simple file compression (using the **Deflate** and **Huffman Coding** algorithms).

**2. User Interface Design**

The interface will be **menu-driven** and will run in a terminal or command-line window. The user will interact with the program by selecting options through the menu.

**Main Menu Flow:**

* **Start** → **Main Menu**
* **Main Menu** will present the user with several options for performing actions on files.
* After completing a task, the user will return to the **Main Menu** until they decide to exit.



**3. Functionalities and Features**

The system will have the following core functionalities:

**A. File Operations:**

1. **Create File**:
   * Users can create a new file and add content to it.
   * The system will prompt for a filename and ensure it doesn't already exist to avoid overwriting.
2. **Edit File**:
   * Users can open an existing file and edit its contents.
   * The user can either replace the file's content or append new content to the file.
3. **Rename File**:
   * Users can rename an existing file.
   * The system will check for file name collisions and ensure the name is valid.
4. **Delete File**:
   * Users can delete a file, with a confirmation prompt to prevent accidental deletion.
5. **View All Files**:
   * Users can view a list of all files in the current directory, displaying their names, sizes, and last modified date.

**B. File Search:**

* Users can search for files by their **filename** or by searching for specific content within the files.

**C. File Information:**

* Users can display file properties such as:
  + **File Name**
  + **File Size**
  + **Creation Date**
  + **Last Modified Date**

**D. Compression Operations:**

1. **Zip File**:
   * Compress a file using the **Deflate Algorithm** (a combination of **Huffman Coding** and **LZ77**).
2. **Unzip File**:
   * Decompress a zip file back to its original state using the **Inflate Algorithm** (the inverse of Deflate).

**E. Exit:**

* Exits the program.

**4. Algorithm Design**

**A. File Compression (Deflate Algorithm)**

* **Huffman Coding**: Used to create a frequency table and then build a binary tree for encoding file data.
* **LZ77 Compression**: Uses sliding windows to replace repeated strings with references to earlier parts of the data.

**B. File Decompression (Inflate Algorithm)**

* **Huffman Decoding**: Reverses the Huffman coding process to restore the original data.
* **LZ77 Decompression**: Restores the file by reversing the sliding window references.

**C. File Handling in C**

* File operations will use the standard **C library** functions such as:
  + fopen() for opening files.
  + fread() and fwrite() for reading and writing data.
  + fclose() for closing the file.
  + remove() for deleting files.

**D. Searching Files**

* Searching will be done by scanning the file contents line by line or using file metadata like name or modification time.
  + - 1. **Error Handling and Validation**

The system will include error handling for:

* Invalid file names (e.g., names with special characters).
* File not found (when editing, renaming, or deleting).
* File access errors (e.g., permission issues).
* Compression or decompression errors.
* User input validation (to ensure valid menu selection).

**IMPLEMENTATION DETAILS**

In ZipWizard, the project is structured to separate the functionality into individual C files and header files for modularity, reusability, and easier maintenance. The header files (.h) define the function prototypes, data structures, and constants, while the C files (.c) contain the implementation of these functions and other logic.

The **ZipWizard** project follows a modular architecture, where different tasks such as file compression, decompression, renaming, searching, editing, and utility functions are handled by individual modules. This modular approach ensures that each component is self-contained, making it easier to maintain, debug, and extend the functionality in the future.

Here’s an overview of how the modules come together in the **ZipWizard** project:

**1. Main Program (main.c)**

This is the entry point where the user interacts with the program. It prompts the user for input and invokes the appropriate functions from the different modules (e.g., compression, decompression, renaming, etc.).

**Responsibilities:**

* Displays the menu to the user.
* Collects user input (e.g., which action to perform).
* Calls functions from the various modules (Compression, Decompression, File Editing, etc.) based on the user's choices.

**2. Compression Module (zipFile.c)**

This module contains the functionality for compressing files. It uses algorithms **LZ77** to convert files into a compressed format.

**Key Functions:**

* **lz77Compress()**: Implements the LZ77 compression algorithm. It reads the input file, identifies repeated substrings, and compresses the data using a sliding window mechanism.
* **zipFile()**: This function can be used to handle the compression process, including writing the compressed data to a new file.

**3. Decompression Module (unzipFile.c)**

Handles the decompression of files that were compressed by the **zipFile.c** module.

**Key Functions:**

* **lz77Decompress()**: This function decompresses files using the LZ77 algorithm, where compressed data (offsets, lengths, characters) is processed and the original file is reconstructed.
* **unzipFile()**: This function is used to handle the decompression process, reading the compressed file and writing the decompressed output.

**4. File Editing Module (editFile.c)**

This module provides the functionality for users to interact with text files. It allows users to open, modify, and save changes to files. It supports appending data to files and reading from them.

**Key Functions:**

* **editFile()**: This function allows users to open a file and make changes to it, such as adding new content.

**5. File Search Module (searchFile.c)**

This module allows users to search for files by their name or keyword. The user can search for files in the current directory that match a given pattern or contain a specific keyword.

**Key Functions:**

* **searchfile()**: Allows the user to search for files by name or by keyword. It uses the Windows API (FindFirstFile and FindNextFile) to search for files matching the specified criteria.

**6. File Rename Module (renameFile.c)**

This module provides functionality for renaming files. It checks if the file exists and ensures that the new name follows specific rules (e.g., file names should end with .txt).

**Key Functions:**

* **renamefile()**: Prompts the user for a file name to rename and checks if the file exists. It then asks for a new file name and performs the renaming operation using the rename() function.

**7. Utility Functions Module (Utils.c)**

The **Utils.c** module contains helper functions that are used across the entire application. These functions handle tasks like input validation, printing formatted messages, and basic file operations.

**Key Functions:**

* **zwPrint()**: A utility function for printing messages to the console. It allows the program to output messages with different severity levels (e.g., INFO, ERROR, SUCCESS).
* **terminalSize(int width,int height)**: sets the size of the terminal window and disables resizing or maximizing on a Windows platform.

**8. View File Information Module ( fileInfo.c):** The function relies on Windows-specific API calls (GetFileAttributesEx, GetFullPathName, FileTimeToSystemTime, etc.), and outputs information about the file through the zwPrint() function, which seems to be a custom logging or display function.

**Key Functions:**

* fileInfo(): Prompts the user to enter a file name and then retrieves and displays various attributes and timestamps of the specified file

**9. Error Handling**

Error handling is essential to the **ZipWizard** project. Several layers of validation and checks are used to ensure that each file operation and user input is valid.

**Error Handling Strategies:**

* **File Existence Check**: Each file operation (reading, writing, or editing) first checks if the file exists, if it can be opened, and if the operation succeeds.
* **Input Validation**: User inputs (e.g., file names, content) are validated for correctness before performing operations on them.
* **Error Messages**: If something goes wrong, clear and descriptive error messages are shown to the user. These are formatted using the zwPrint function with appropriate labels for error levels (e.g., ERROR\_FILE, INFO, SUCCESS).

**10. File Handling**

File handling is performed carefully to ensure that the program does not cause data corruption or loss during its operations. Various checks are performed during file reading and writing operations.

**File Operations:**

* **Reading**: Files are opened using fopen() in appropriate modes (r for reading, a for appending, etc.). We also make sure to close files using fclose() to free resources.
* **Writing**: Data is written to files using fprintf() for appending content. We validate the file paths and names before attempting to write to them.
* **Buffered I/O**: When dealing with large files, fgets() and other buffered reading functions are used to handle the input safely.

**11. Algorithm Choice**

The core algorithm used in **ZipWizard** is the **LZ77** compression algorithm, which is effective for compressing text-based files. It works by finding repeated substrings and encoding them as references to previous occurrences within a sliding window of the data.

**LZ77 Compression:**

* **Sliding Window Mechanism**: The algorithm uses a window to store previously seen data. If the same sequence of characters is found within the window, the data is replaced with a reference to the earlier part of the string (offset and length).

**File Structure:**

* The compressed files store metadata such as the dictionary and offsets. This metadata helps in efficient decompression.

**12. Design Patterns**

The **ZipWizard** project adheres to several key design principles:

**Modularity:**

Each task (compressing, decompressing, editing) is handled by its own dedicated module. This makes the code more maintainable and reusable.

**Separation of Concerns:**

Each module is responsible for a specific task. This allows modifications in one area (e.g., compression) without affecting other parts of the program (e.g., file editing).

**Error Handling Strategy:**

The program uses early returns in case of errors, which keeps the control flow clean and avoids deep nesting of conditional statements.

**13. User Interface**

The user interface is text-based, designed for simplicity and ease of interaction. Users are presented with options through prompts and asked to provide the necessary inputs (file names, keywords, etc.).

**User Interaction:**

* **Prompts**: The program uses clear and concise prompts to guide the user through various operations.
* **Feedback**: The program provides informative messages after each operation (e.g., success, error messages).

**Overview of LZ77 Compression Technique**

LZ77 (Lempel-Ziv 1977) is a lossless data compression algorithm that works by replacing repeated occurrences of data with references to a single copy of that data. It's based on the idea of finding repeated sequences (strings) within the data and replacing them with a reference to the previous occurrence. This technique is widely used in file compression formats, including .zip files, and is the foundation of algorithms like ZIP, GZIP, and PNG.

The **basic steps of the LZ77 algorithm** are:

1. **Sliding Window**: LZ77 uses a sliding window approach where a "window" of fixed size is maintained over the input data. This window is split into two parts: a search buffer (containing previously seen data) and a look-ahead buffer (which contains the upcoming data to be compressed).
2. **Matching**: As the look-ahead buffer is examined, LZ77 tries to find the longest match of a sequence in the search buffer.
3. **Encoding**: When a match is found, LZ77 encodes this match as a triplet: (offset, length, nextChar):
   * **offset**: The position of the matched string in the search buffer.
   * **length**: The length of the match.
   * **nextChar**: The next character in the data that follows the matched string (i.e., the part not covered by the match).
4. **Update**: The sliding window is updated by shifting the window forward by the length of the match and adding the next character from the input.

**Benefits of LZ77:**

1. **Simple and Effective**: LZ77 is relatively simple to implement and provides efficient compression for many types of data.
2. **Lossless Compression**: LZ77 does not lose any data during compression, making it suitable for applications where the original data must be perfectly preserved (e.g., software distribution, file storage).
3. **Widely Used**: LZ77 is the foundation of many popular file compression algorithms (e.g., ZIP, GZIP, PNG).
4. **Adaptability**: LZ77 can handle various types of data, making it effective for a wide range of files, from text to binary.

**Drawbacks of LZ77:**

1. **Limited Compression for Small or Highly Entropic Data**: LZ77 performs poorly on files that do not contain much redundancy, such as highly random or already compressed data. Files with little or no repetition will not benefit much from LZ77 compression.
   * For example, a file consisting of random characters would yield poor compression rates since there are no repeated sequences to exploit.
2. **Sliding Window Size and Memory Usage**: The algorithm requires maintaining a sliding window of a fixed size, which consumes memory. Larger windows may improve compression rates but increase memory usage and computational cost. A larger window may also increase the time required to search for matches.
   * This is especially important for devices with limited memory (e.g., embedded systems, mobile devices).
3. **Compression Efficiency**: LZ77 might not provide the best compression ratio compared to other advanced algorithms like LZMA or BZIP2. These algorithms use more sophisticated techniques to find patterns and reduce file sizes further but at the cost of increased computational complexity.
   * While LZ77 works well for repetitive data, more advanced techniques like **Huffman coding**, **Burrows-Wheeler Transform (BWT)**, or **arithmetic coding** are often used in combination with LZ77 for better performance.
4. **Speed vs. Compression Trade-Off**: The algorithm's performance can be a trade-off between speed and compression ratio. Finding longer matches leads to better compression but increases the time complexity of the algorithm, especially for large files.
   * For real-time applications or streaming data, this might result in slower processing times.
5. **No Adaptive Behavior**: LZ77 operates on a fixed-size window and doesn’t adapt dynamically to different patterns or data structures. This can lead to suboptimal performance when dealing with complex or highly variable data types.
6. **Not Always Optimal for Binary Data**: While LZ77 is effective for text-based data, its performance may degrade when applied to binary or already compressed data files. Other techniques, such as **Huffman coding** or **delta encoding**, may provide better results for specific binary file types.

**Conclusion:**

LZ77 is a straightforward and widely used compression technique that works well for many types of data, especially when there is redundancy in the input. However, it has limitations in its efficiency, particularly when dealing with data that lacks repetition or has complex patterns. Its performance is also dependent on the window size and available memory, and it may not always offer the best compression compared to more sophisticated algorithms like LZMA or BZIP2.

To enhance LZ77, it is often combined with other compression techniques (like Huffman coding or arithmetic coding) to improve compression ratios without sacrificing speed. However, in cases where simplicity and fast decompression are prioritized, LZ77 remains an effective and widely used method.

**BIBLIOGRAPHY**

**Microsoft Developer Network (MSDN)**. (2024). *Windows API Functions*. Microsoft. Retrieved from: <https://docs.microsoft.com/en-us/windows/win32/api/>

* MSDN is the primary resource for Windows system programming. It provides the documentation for functions like SetConsoleWindowInfo, GetConsoleWindow, SetWindowPos, and GetFileAttributesEx, which are used in ZipWizard to manipulate the console window and file attributes.

**Wikipedia Contributors**. (2024). *LZ77 Compression*. Retrieved from: <https://en.wikipedia.org/wiki/LZ77_and_LZ78>

* + This page provides a high-level overview of the LZ77 algorithm, which forms the core of the compression technique likely used in ZipWizard.

**API Documentation**:

* **Microsoft Docs**. (2024). *GetFileAttributesEx Function*. Retrieved from: <https://learn.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-getfileattributesexw>
  + This documentation provides details about the GetFileAttributesEx function, which is used in the ZipWizard project to retrieve file attributes such as creation, modification, and access times.
* **Microsoft Docs**. (2024). *SetConsoleWindowInfo Function*. Retrieved from: <https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-setconsolewindowinfo>
  + This page provides details about the SetConsoleWindowInfo function, used in ZipWizard to control the size of the console window.

**Stack Overflow**. (2024). Questions related to *SetWindowPos* and Windows Console Programming. Retrieved from: <https://stackoverflow.com/questions/>

* + - A community-driven forum where many programming questions related to Windows-specific APIs and console window manipulation can be found.

**REFERENCES**

**Research Papers and Articles**:

* **Huffman, D. A.** (1952). *A Method for the Construction of Minimum-Redundancy Codes*. Proceedings of the IRE, 40(9), 1098–1101. https://doi.org/10.1109/JRPROC.1952.273898
  + This paper introduces the Huffman coding algorithm, which is often used in file compression tasks. Although ZipWizard focuses on LZ77, Huffman coding is also an important concept in compression.
* **Ziv, J., & Lempel, A.** (1977). *A Universal Algorithm for Sequential Data Compression*. IEEE Transactions on Information Theory, 23(3), 337–343. https://doi.org/10.1109/TIT.1977.1055714
  + This paper presents the LZ77 algorithm for data compression, which serves as a foundation for the compression techniques in ZipWizard.