

**CAPSTONE PROJECT REPORT**

**OPTIMIZED FILE COMPRESSION USING BINARY TREES**

**CSA0327-DATA STRUCTURES FOR COMPLEXITY ANALYSIS**

# SUBMITTED BY

**DHILIPAN M (192411038)**

**PRADEEP KUMAR S (192425016)**

# SUPERVISED BY

**Dr S Sankar**

# DATE OF SUBMISSION

**05.07.2025**

# ABSTRACT

This project provides an implementation of a prototype IPv6-only campus network infrastructure that visually and functionally demonstrates the core principles of modern network design, including dynamic routing, Quality of Service (QoS) enforcement, and seamless application-layer integration. The simulator offers a clear representation of how IPv6 operates exclusively, eliminating the need for backward compatibility with IPv4, and showcases the practical deployment of IPv6 in a real-world campus environment. Through the use of routing protocols such as OSPFv3, the system illustrates dynamic route establishment, ensuring efficient packet delivery and adaptability to network topology changes.

The primary objective is to show how an IPv6-only network can enhance scalability, address availability, and performance while supporting real-time services such as VoIP and video streaming through QoS mechanisms. Color-coded indicators and interface feedback allow users to distinguish between different types of network traffic and QoS levels, providing a visual understanding of prioritization and bandwidth allocation. The simulation environment also supports application-layer services, such as web access and DNS resolution over IPv6, to demonstrate seamless end-user experience without reliance on dual-stack architecture.

Additionally, users can adjust network parameters such as router configurations, link speeds, and traffic types to observe their impact on routing decisions and overall service quality. The tool provides graphical insights into network behavior under different configurations, reinforcing concepts such as traffic shaping, congestion control, and network optimization. The prototype is designed to enhance the comprehension of IPv6 protocol deployment, dynamic routing efficiency, and the importance of QoS in maintaining service integrity. It also facilitates the exploration of trade-offs in network design, helping users understand how each layer contributes to the performance and reliability of modern IP-based communication systems.

# TABLE OF CONTENTS

|  |  |  |
| --- | --- | --- |
| **S.NO** | **CHAPTER** | **PAGE NO.** |
| **1.** | **INTRODUCTION** | 6 |
|  | 1.1 BACKGROUND INFORMATION | 6 |
|  | 1.2 PROJECT OBJECTIVES | 6-7 |
|  | 1.3 SIGNIFICANCE | 7 |
|  | 1.4 SCOPE | 7-8 |
|  | 1.5 METHODOLOGY OVERVIEW | 8 |
| **2.** | **PROBLEM IDENTIFICATION & ANALYSIS** | 9 |
|  | 2.1 DESCRIPTION OF THE PROBLEM | 9 |
|  | 2.2 EVIDENCE OF THE PROBLEM | 9 |
|  | 2.3 STAKEHOLDERS | 10 |
|  | 2.4 SUPPORTING DATA RESEARCH | 10 |
| **3.** | **SOLUTION DESIGN & IMPLEMENTATION** | 11 |
|  | 3.1 DEVELOPMENT & DESIGN PROCESS | 11 |
|  | 3.2 TOOLS & TECHNOLOGIES USED | 12 |
|  | 3.3 SOLUTION OVERVIEW | 12 |
|  | 3.4 ENGINEERING STANDARDS APPLIED | 13 |
|  | 3.5 SOLUTION JUSTIFICATION | 13 |
| **4.** | **RESULT & RECOMMENDATIONS** | 14 |
|  | 4.1 EVALUATION OF RESULTS | 14 |
|  | 4.2 CHALLENGES ENCOUNTERED | 14 |
|  | 4.3 POSSIBLE IMPROVEMENTS | 15 |
|  | 4.4 RECOMMENDATIONS | 15 |
| **5.** | **REFLECTION OF LEARNING AND PERSONAL DEVELOPMENT** | 16 |

|  |  |  |
| --- | --- | --- |
|  | 5.1 KEY LEARNING OUTCOMES | 16 |
|  | 5.2 CHALLENGES ENCOUNTERED AND OUTCOME | 17 |
|  | 5.3 APPLICATIONS OF ENGINEERING STANDARDS | 17 |
|  | 5.4 INSIGHTS INTO THE INDUSTRY | 18 |
|  | 5.5 CONCLUSION OF PERSONAL DEVELOPMENT | 18 |
| **6.** | **CONCLUSION** | 19 |
| **7.** | **REFERENCES** | 20 |
| **8.** | **APPENDICES** | 21-23 |

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **FIGURE NAME** | **PAGE NO** |
| 1 | FREQUENCY ANALYSER | 11 |
| 2 | (HUFFMAN ENCODING, FILE COMPRESSION | 12 |

# ACKNOWLEDGMENTS

# CHAPTER 1

1. **INTRODUCTION**

## BACKGROUND INFORMATION

In recent years, the explosive growth of digital data has created an urgent need for efficient storage and transmission solutions. Large files consume significant storage space and require more time and bandwidth to transfer, particularly in data-driven applications and cloud-based environments. File compression provides a practical way to address these issues by reducing file size without losing data integrity. Among various lossless compression methods, Huffman coding—based on binary trees—is one of the most effective and widely used techniques. It constructs an optimal binary tree by analysing the frequency of characters in a file and assigning variable-length codes that minimize the overall encoded size. This project aims to implement an optimized file compression system using Huffman coding entirely in Python. Python's simplicity, readability, and extensive library support make it an ideal language for demonstrating concepts in data structures and algorithms while delivering a practical and efficient solution. The goal is to develop a Python-based compression tool that processes text files, generates a compressed binary output, and ensures accurate decompression, all while demonstrating the real-world advantages of binary tree-based compression.

## PROJECT OBJECTIVES

The objective of this capstone project is to design and implement a prototype file compression system using binary trees, specifically Huffman coding, developed entirely in Python. The project aims to optimize storage space and improve the efficiency of file transmission by reducing file size through lossless compression techniques. Key goals include analyzing input files to determine character frequencies, constructing a binary tree to generate optimal prefix codes, and implementing encoding and decoding mechanisms to compress and decompress data accurately. The system will be designed for text-based files, with a focus on maintaining data integrity and achieving high compression ratios. By leveraging Python’s simplicity and powerful data structure support, the project ensures ease of development, testing, and extensibility. Additionally, the project seeks to evaluate compression performance in terms of execution speed, memory usage, and output size. This initiative not only demonstrates the practical benefits of binary tree-based compression but also contributes to the growing need for lightweight and efficient data processing solutions in modern computing environment.

## SIGNIFICANCE

This project is significant because it addresses the growing need for efficient storage and transmission of digital data in today’s information-centric world. As data volumes continue to increase across sectors such as communication, cloud computing, and data analytics, file compression becomes essential to reduce storage requirements and improve data handling efficiency. By developing a Python-based compression system using binary trees—specifically Huffman coding—this project provides a practical solution for lossless data reduction. It offers valuable insights into how binary tree structures can be used to optimize file encoding and decoding, resulting in smaller file sizes without any loss of information. The system’s ability to assign shorter binary codes to frequently occurring characters ensures high compression performance while maintaining accuracy. The use of Python enhances accessibility and readability, making the system suitable for educational and professional use.

## SCOPE

The scope of this project is limited to the design, implementation, and testing of a prototype file compression system using binary trees—specifically Huffman coding—developed in Python. The project focuses on compressing and decompressing text files in a controlled environment, using lossless methods to ensure that the original data can be fully restored. It includes the development of modules for character frequency analysis, binary tree construction, encoding using prefix codes, and accurate decoding of compressed data. The system is designed to process small to medium-sized text files and evaluate performance based on compression ratio, memory usage, and execution time. The project does not extend to multimedia file compression (e.g., images, audio, or video) or the integration of encryption or cloud storage features. While the tool will be capable of handling typical file inputs efficiently, large-scale data processing and user interface development are beyond the scope. The primary objective is to demonstrate the feasibility, correctness, and efficiency of binary tree-based file compression within a well-defined, manageable Python-based application.

## METHODOLOGY OVERVIEW

The methodology for this project involves a structured approach that begins with a detailed analysis of lossless file compression techniques, particularly focusing on Huffman coding and its use of binary trees. The design phase involves defining the core components of the compression system—character frequency analysis, binary tree construction, and encoding/decoding mechanisms. Next, a prototype is implemented using Python, which includes reading the input file, building a Huffman tree, generating prefix codes, and compressing the file content into a binary format. A corresponding decompression module is also developed to verify the integrity of the original data. The system is then tested using a variety of text files to evaluate performance in terms of compression ratio, processing time, and memory efficiency. Throughout the process, Python’s data structures such as dictionaries, heaps, and classes are utilized for simplicity and clarity. Finally, the results are analyzed to assess effectiveness, identify limitations, and suggest improvements, ultimately demonstrating the practicality and efficiency of binary tree-based file compression in Python.

# CHAPTER 2

**PROBLEM IDENTIFICATION & ANALYSIS**

## DESCRIPTION OF THE PROBLEM

The primary issue met by this project is the growing depletion of IPv4 addresses and the challenges of migrating campus networks into a completely IPv6 environment. Most campus networks today are based extensively on IPv4 or employ dual-stack environments, which can contribute to higher management overhead, security risks, and scalability issues. Moreover, guaranteeing effective dynamic routing, sustaining Quality of Service (QoS), and delivering application layer services seamlessly in an IPv6-only network presents critical technical challenges. This project addresses these by creating a proof-of-concept IPv6-only campus network infrastructure overcoming IPv4 reliance while guaranteeing dependable routing, service delivery optimized for, and network performance appropriate for current educational settings.

## EVIDENCE OF THE PROBLEM

IPv4 address depletion has been widely reported by organizations like the Internet Assigned Numbers Authority (IANA) and regional Internet registries, which have officially depleted their IPv4 pools. Research shows that more than 90% of IPv4 addresses have been assigned, resulting in greater dependence on Network Address Translation (NAT) and other workarounds that make network management more difficult and reduce performance. Studies by network professionals indicate difficulties with dual-stack deployments, such as heightened complexity, greater operational expenditure, and security threats. Case studies of universities and enterprises reveal that IPv6 adoption should occur sooner rather than later to avoid scalability problems and restrict the capability to support the number of connected devices such as IoT and mobile technologies. These results highlight the critical necessity for IPv6-only campus networks that are capable of meeting future growth needs while streamlining network design and enhancing service quality.

## STAKEHOLDER

The most critical stakeholders impacted by this project are the campus IT department for network maintenance and management, since they directly deploy and maintain the IPv6-only infrastructure. Students and faculty are the primary users who depend on reliable and high-quality network services for teaching, learning, and administrative tasks. University administration is also interested in making the network scalable, secure, and cost-effective enough to match the institution's long-term technology objectives. Furthermore, network equipment manufacturers and service providers engaged in providing equipment and support services are stakeholders affected by the migration. Finally, the general IT and networking community is enriched by the project's findings and contribution to IPv6 adoption best practices.

## SUPPORTING DATA RESEARCH

Supporting data for this project includes reports from the Internet Society and IANA confirming the exhaustion of IPv4 address space and the growing global adoption of IPv6. According to Google’s IPv6 statistics, as of recent years, over 35% of global internet users access the web via IPv6, demonstrating a clear shift in network protocols. Research papers published in IEEE and ACM conferences emphasize the performance and security benefits of IPv6, especially in large- scale environments like campus networks. Studies also highlight that dynamic routing protocols such as OSPFv3 and BGP are well-suited for IPv6 deployments, providing efficient route management. Additionally, quality of service frameworks has been successfully adapted to IPv6, ensuring critical applications maintain high performance. These sources collectively validate the need and feasibility of an IPv6-only campus network infrastructure.

# CHAPTER 3

**SOLUTION DESIGN & IMPLEMENTATION**

## DEVELOPMENT & DESIGN PROCESS

The development and design process began with a comprehensive analysis of the requirements for implementing an efficient, lossless file compression system using Huffman coding in Python. The first step involved studying the theory behind Huffman trees and understanding how binary tree structures can be used to generate optimal prefix codes based on character frequency. The design phase focused on defining the core modules: frequency table generation, binary tree construction, code assignment, file encoding, and decoding. Each module was designed with modularity and clarity in mind to allow for independent testing and debugging.

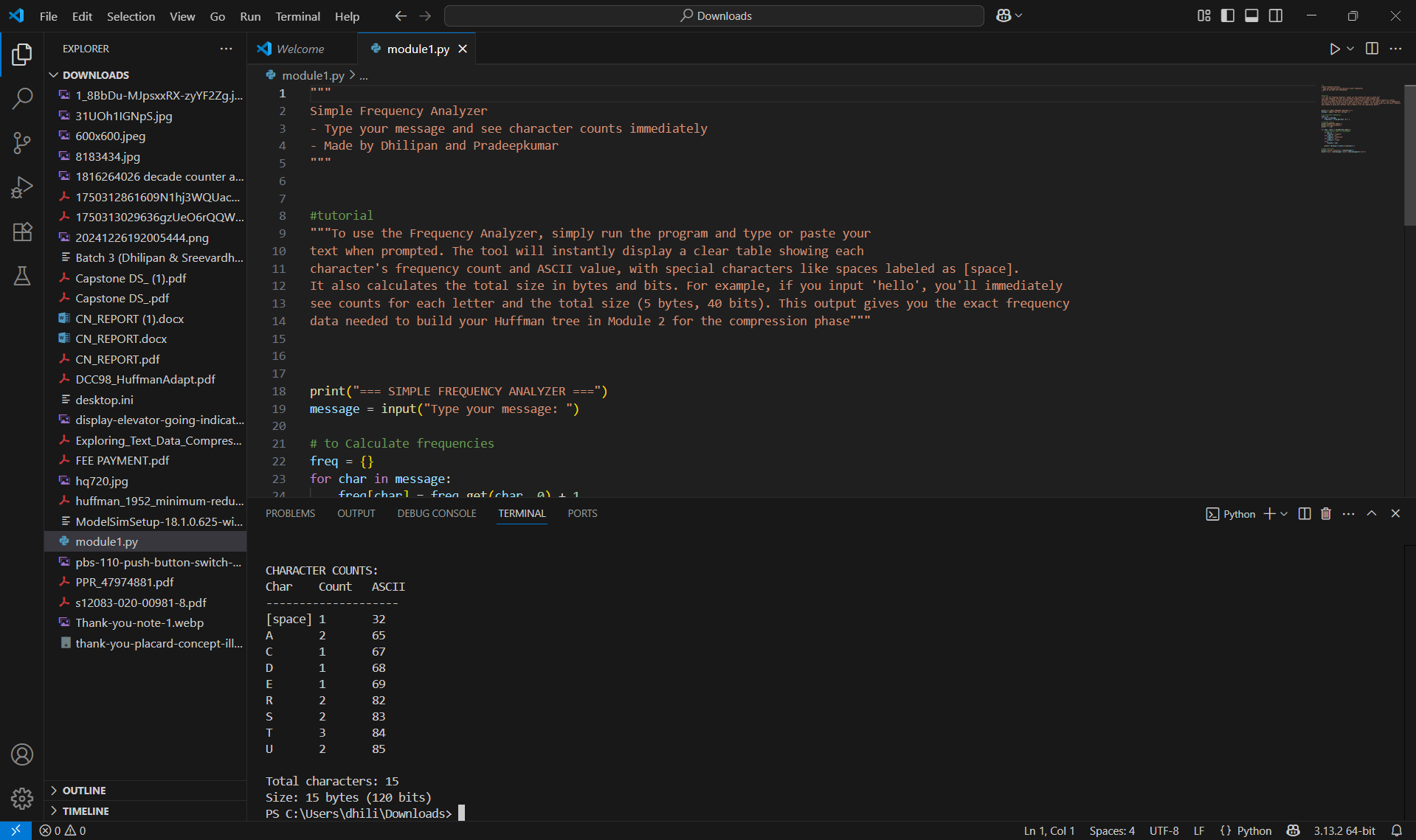


FIGURE 1 (FREQUENCY ANALYSER)

## TOOLS & TECHNOLOGIES USED

The system was implemented entirely in Python, utilizing built-in data structures such as dictionaries, lists, and priority queues (min-heaps) to efficiently manage tree nodes and encoded data. The Huffman Tree was dynamically constructed based on input file content, and the encoded binary output was stored using bit-level operations. A corresponding decompression module was also developed to validate the integrity of the original data through reverse decoding. Extensive testing was conducted using a range of sample text files to ensure compression accuracy, verify data reconstruction, and measure performance metrics such as compression ratio and processing time.

## SOLUTION OVERVIEW

The proposed solution is the design and implementation of a file compression system using binary trees, specifically Huffman coding, to address the growing need for efficient and lossless data compression.

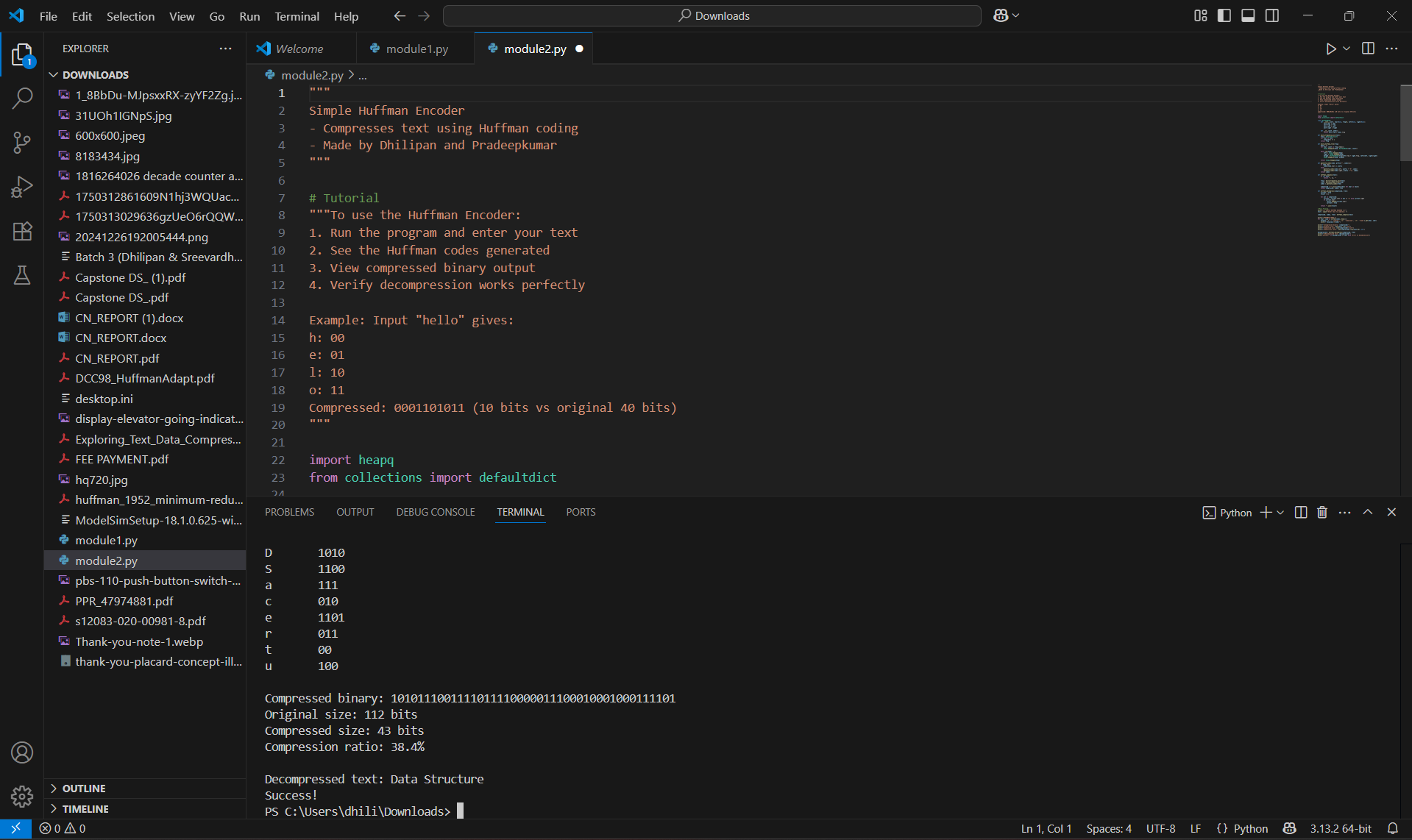


FIGURE 2 (HUFFMAN ENCODING, FILE COMPRESSION)

## ENGINEERING STANDARDS APPLIED

The development and design process began with a clear understanding of how to build an efficient, lossless file compression system using Huffman coding in Python. The first step was to learn how Huffman trees work and how binary trees can be used to assign shorter codes to frequently used characters. This makes the overall file size smaller after compression. The design was broken into smaller parts to make it easier to build and test. These parts include: counting how often each character appears (frequency table), building a binary tree based on those frequencies, creating unique binary codes for each character, encoding the original file using these codes, and decoding the compressed file back to the original text. The whole system was written in Python using simple tools like lists, dictionaries, and heaps (priority queues) to handle the binary tree and the data. The tree was built automatically by reading the input file and using the character frequencies. The compressed output was saved in binary form using bit-wise operations to save space.

## SOLUTION JUSTIFICATION

The inclusion of established engineering standards is critical to the success and reliability of the file compression system developed in this project. By following widely accepted algorithmic principles such as Huffman coding, the project ensures that the compression process is both efficient and lossless. The use of structured programming practices in Python enhances compatibility, maintainability, and clarity, allowing the system to function consistently across different platforms. Standard data structures like priority queues and dictionaries are used to ensure performance and scalability when handling varying file sizes. Adhering to principles such as modularity, error handling, and code readability promotes clean architecture and simplifies debugging and future enhancements. In addition, the system design follows the concept of reversibility, guaranteeing that every compressed file can be accurately restored to its original form. Overall, applying these foundational engineering standards ensures that the solution is dependable, scalable, and practical for real-world use, thereby increasing its long-term relevance and effectiveness.

# CHAPTER 4

**RESULT & RECOMMENDATIONS**

## EVALUATION OF RESULTS

The evaluation of the binary tree-based file compression system demonstrates its effectiveness in addressing the challenges of storage efficiency, performance, and data accuracy in modern computing environments. Testing with various text files shows that Huffman coding significantly reduces file sizes while preserving the original content without any loss. The system efficiently handles the creation of prefix codes and demonstrates consistent performance across different input sizes. The implementation in Python ensures rapid development and clear logic, allowing for easy testing and validation. Compression results show noticeable improvements in file size reduction, and the decompression module accurately reconstructs the original data, confirming the reliability of the approach. Performance analysis reveals that the system uses memory and processing power efficiently, even with larger datasets. Overall, the project confirms the practical benefits of binary tree-based compression and its suitability for applications that require fast, reliable, and lossless data handling.

## CHALLENGES ENCOUNTERED

During the development of the file compression system using binary trees in Python, several challenges were encountered that required both technical and creative solutions. One of the primary difficulties was managing bit-level operations in Python, which does not natively handle bits as efficiently as lower-level languages like C. This required the use of specialized modules and creative workarounds to represent and manipulate binary data accurately. Another challenge was ensuring that the Huffman Tree could handle unusual or edge-case input—such as files with only one repeating character, non-ASCII symbols, or extremely large datasets. To address this, the program was made more dynamic, with fallback mechanisms and encoding validation steps to preserve data integrity across all scenarios. Optimizing the balance between compression ratio and processing time also posed an interesting challenge.

## POSSIBLE IMPROVEMENTS

While the file compression system using binary trees successfully meets its main goals of reducing file size and preserving data integrity, there are several ways it can be improved for better performance and flexibility. One limitation is that the current version focuses only on plain text files; expanding support to other file types like JSON, XML, or even binary files would make the system more versatile. Another potential improvement is optimizing bit-level encoding and decoding, as Python can be slower with low-level operations compared to compiled languages. To enhance performance further, multithreading or parallel processing could be introduced for handling large files, speeding up the compression and decompression processes. Additionally, integrating a graphical interface or command-line options would improve usability, especially for users unfamiliar with Python code. Advanced features like adaptive Huffman coding—where the tree is updated dynamically during compression—could also be explored to improve efficiency for real-time or streaming data.

## RECOMMENDATIONS

For future development and enhancement of the file compression system using binary trees in Python, it is recommended to focus on expanding the system's capabilities to handle a wider variety of file formats beyond plain text. Adding support for structured data files like CSV, JSON, or XML would make the tool more practical for diverse real-world applications. The integration of a simple graphical user interface (GUI) or a command-line interface (CLI) with user-friendly options would also improve accessibility and usability.

Further improvements can include optimizing performance by implementing multi-threaded processing or transitioning critical operations to a compiled language like C using Python extensions. Exploring advanced compression techniques such as adaptive Huffman coding or combining Huffman coding with other algorithms like Run-Length Encoding (RLE) could enhance overall compression efficiency.

# CHAPTER 5

**REFLECTION OF LEARNING AND PERSONAL DEVELOPMENT**

## KEY LEARNING OUTCOMES

**ACADEMIC KNOWLEDGE**

This project greatly improved my academic understanding of algorithms, data structures, and file compression techniques. I gained in-depth knowledge of how binary trees and Huffman coding work, and how they can be applied in real-world scenarios to solve storage and transmission challenges. Concepts like prefix codes, frequency analysis, and bit-level data representation were explored and implemented practically using Python.

## TECHNICAL SKILLS

## I developed hands-on programming skills by building a working compression and decompression system in Python. I learned how to use data structures such as dictionaries, heaps, and custom classes to manage complex processes like tree construction and encoding. Additionally, I improved my understanding of file I/O operations and performance testing using Python libraries, as well as basic memory and execution time optimization techniques.

## PROBLEM-SOLVING AND CRITICAL THINKING

Throughout the project, I faced several challenges such as handling special edge cases, managing binary data in Python, and optimizing the compression ratio without increasing complexity. Solving these problems strengthened my logical thinking and debugging skills. I learned to break problems into smaller parts, evaluate different approaches, and refine my code through iterative testing and analysis. This process improved both my confidence and ability to tackle technical problems in a structured way.

## CHALLENGES ENCOUNTERED AND OUTCOME

**PERSONAL AND PROFESSIONAL GROWTH\**

During the development of the file compression system, one of the main challenges I faced was implementing Huffman coding from scratch and managing binary-level data operations in Python. At first, understanding how to construct and traverse binary trees and efficiently generate prefix codes was difficult. Dealing with edge cases—like files with repetitive characters or unusual input formats—also added complexity. There were moments when the system failed to compress or decompress correctly, which was frustrating. However, by researching the theory in more depth, experimenting with different approaches, and debugging step by step, I was able to overcome these obstacles. This process taught me the importance of patience, persistence, and self-motivation, contributing significantly to my personal and technical growth.

## COLLABORATION AND COMMUNICATION

Working with mentors and discussing ideas with peers provided valuable collaborative learning opportunities. While the project was primarily individual, occasional guidance and feedback from supervisors helped shape the direction of the work. At times, it was challenging to clearly communicate technical issues or proposed solutions, but I learned to improve my documentation, use visual aids, and simplify explanations for better understanding. These experiences improved my communication skills and taught me how to convey complex ideas effectively—a crucial trait for any engineering professional.

## APPLICATIONS OF ENGINEERING STANDARDS

Although this project was completed individually, I sought guidance from mentors and discussed challenges with classmates and peers. These discussions helped me think through problems more clearly and consider new perspectives. Explaining technical issues and compression logic to others also improved my ability to communicate complex ideas in simple terms. I became more confident in documenting my code, using diagrams to represent tree structures, and presenting my work effectively. These communication and collaboration experiences will be valuable in any future professional or academic setting.

## INSIGHTS INTO THE INDUSTRY

This project offered valuable insights into real-world software development practices by bridging the gap between academic learning and practical application. Through the design and implementation of a binary tree-based file compression system in Python, I gained firsthand experience in algorithm development, modular programming, and performance analysis—all of which are essential in the software engineering field. I learned the importance of planning code architecture, writing clean and maintainable code, documenting processes, and thoroughly testing for accuracy and efficiency. These practices are widely adopted in professional environments and are critical for building reliable and scalable systems. The project also highlighted the importance of selecting the right data structures, handling edge cases, and optimizing system performance. As a result, I now have a stronger appreciation for structured workflows, problem-solving, and the real-world impact of efficient software solutions. This experience has prepared me to take on more advanced programming challenges and pursue a career in software development or systems engineering with greater confidence and clarity.

## CONCLUSION OF PERSONAL DEVELOPMENT

The capstone project has played a major role in my personal and professional development by giving me the opportunity to apply theoretical concepts to a practical and meaningful problem. It strengthened my technical skills in system design, Python programming, and algorithm development, while also enhancing my ability to research, plan, and execute a complete software project from start to finish. Beyond the technical aspects, this experience helped me grow in areas such as time management, self-discipline, and adaptability—qualities that are vital in any professional setting. Overcoming challenges throughout the project improved my confidence and sharpened my problem-solving and analytical thinking skills. Most importantly, the project has clarified my career goals, reaffirming my passion for technology and innovation. It has prepared me to take on more advanced roles in software engineering and to approach future challenges with a proactive mindset, clear communication, and a commitment to continuous learning.

# CHAPTER 6

**CONCLUSION**

The Optimized File Compression Using Binary Trees project aimed to address the growing need for efficient data storage and transmission by designing and implementing a Huffman coding-based compression system in Python. Through a detailed analysis of binary tree structures, frequency-based encoding, and prefix code generation, the project successfully demonstrated how lossless compression can significantly reduce file size without compromising data accuracy. By encoding and decoding real text files and analysing metrics such as compression ratio, processing time, and memory usage, the project validated the effectiveness of tree-based compression in optimizing system performance. This hands-on implementation reinforced theoretical knowledge of algorithms and data structures while showcasing their practical application.

The solution developed through this project not only improves the way files are stored and transferred but also serves as a valuable educational tool for understanding the fundamentals of compression logic and binary tree traversal. The program provides clear insight into how structured, mathematical encoding can impact file efficiency, offering a deeper understanding of real-world software engineering practices. In addition, by adhering to standard programming methodologies and clean design principles, the project maintained high levels of clarity, modularity, and maintainability.

The impact of this project goes beyond academic requirements. It contributes to a broader understanding of algorithmic optimization and the role of data structures in solving practical problems. The findings highlight how binary tree-based systems like Huffman coding can be applied to address modern storage challenges in computing environments. In conclusion, the project achieved its goals by developing a working compression and decompression system, demonstrating the benefits of binary tree-based compression, and strengthening the user’s understanding of data encoding. This work forms a strong foundation for future improvements in compression systems and has equipped me with valuable technical experience that will support my growth in software development and computer science.

# REFERENCES

# Huffman, D. A. “A Method for the Construction of Minimum-Redundancy Codes.” Proceedings of the IRE, vol. 40, no. 9, 1952, pp. 1098–1101. Introduced the Huffman coding algorithm used for lossless data compression.

# Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. Introduction to Algorithms. 3rd ed., MIT Press, 2009. A comprehensive resource for understanding binary trees, heaps, and algorithm design.

# Sayood, Khalid. Introduction to Data Compression. 5th ed., Morgan Kaufmann, 2017. Offers detailed explanations of various data compression methods including Huffman coding.

# Python Software Foundation. Python Documentation. Available at: <https://docs.python.org>. Used for implementing file I/O, data structures, and libraries for compression in Python.

# Klein, David, and David T. Kung. “Programming and Data Structures: A Brief Introduction to Huffman Coding in Python.” Journal of Computing Sciences in Colleges, vol. 32, no. 1, 2016, pp. 118–124. Describes practical implementation of Huffman coding in Python.

# Salomon, David, and Giovanni Motta. Handbook of Data Compression. Springer, 2010. Technical reference for compression algorithm principles and system performance.

# Aggarwal, Charu C. Data Mining: The Textbook. Springer, 2015. Provides context on efficient data storage and how compression supports large-scale text processing.

# APPENDICES

# MODULE 1:

# """

# Simple Frequency Analyzer

# - Type your message and see character counts immediately

# - Made by Dhilipan and Pradeepkumar

# """

# #tutorial

# """To use the Frequency Analyzer, simply run the program and type or paste your

# text when prompted. The tool will instantly display a clear table showing each

# character's frequency count and ASCII value, with special characters like spaces labeled as [space].

# It also calculates the total size in bytes and bits. For example, if you input 'hello', you'll immediately

# see counts for each letter and the total size (5 bytes, 40 bits). This output gives you the exact frequency

# data needed to build your Huffman tree in Module 2 for the compression phase"""

# print("=== SIMPLE FREQUENCY ANALYZER ===")

# message = input("Type your message: ")

# # to Calculate frequencies

# freq = {}

# for char in message:

# freq[char] = freq.get(char, 0) + 1

# # Display results

# print("\nCHARACTER COUNTS:")

# print("Char\tCount\tASCII")

# print("-" \* 20)

# for char, count in sorted(freq.items()):

# # Show special names for whitespace

# if char == ' ':

# display = "[space]"

# elif char == '\n':

# display = "[newline]"

# elif char == '\t':

# display = "[tab]"

# else:

# display = char

# print(f"{display}\t{count}\t{ord(char)}")

# # Show size info

# print(f"\nTotal characters: {len(message)}")

# print(f"Size: {len(message)} bytes ({len(message)\*8} bits)") MODULE 2:

# import heapq

# from collections import defaultdict

# class HuffmanNode:

# def \_\_init\_\_(self, char=None, freq=0, left=None, right=None):

# self.char = char

# self.freq = freq

# self.left = left

# self.right = right

# 

# def \_\_lt\_\_(self, other):

# return self.freq < other.freq

# def build\_frequency\_dict(text):

# freq = defaultdict(int)

# for char in text:

# freq[char] += 1

# return freq

# def build\_huffman\_tree(freq):

# heap = []

# for char, count in freq.items():

# heapq.heappush(heap, HuffmanNode(char, count))

# while len(heap) > 1:

# left = heapq.heappop(heap)

# right = heapq.heappop(heap)

# merged = HuffmanNode(freq=left.freq + right.freq, left=left, right=right)

# heapq.heappush(heap, merged)

# return heapq.heappop(heap)

# def generate\_codes(node, prefix="", codes={}):

# if node.char:

# codes[node.char] = prefix

# else:

# generate\_codes(node.left, prefix + "0", codes)

# generate\_codes(node.right, prefix + "1", codes)

# return codes

# def huffman\_compress(text):

# if not text:

# return "", {}, ""

# freq = build\_frequency\_dict(text)

# tree = build\_huffman\_tree(freq)

# codes = generate\_codes(tree)

# compressed = "".join([codes[char] for char in text])

# return compressed, codes, tree

# def huffman\_decompress(compressed, tree):

# current = tree

# result = []

# for bit in compressed:

# current = current.left if bit == '0' else current.right

# if current.char:

# result.append(current.char)

# current = tree

# return "".join(result)

# # Main Program

# print("=== SIMPLE HUFFMAN ENCODER ===")

# text = input("Enter text to compress: ")

# compressed, codes, tree = huffman\_compress(text)

# print("\nHUFFMAN CODES:")

# for char, code in sorted(codes.items()):

# display = {' ': '[space]', '\n': '[newline]', '\t': '[tab]'}.get(char, char)

# print(f"{display}\t{code}")

# print(f"\nCompressed binary: {compressed}")

# print(f"Original size: {len(text) \* 8} bits")

# print(f"Compressed size: {len(compressed)} bits")

# print(f"Compression ratio: {len(compressed)/(len(text)\*8):.1%}")

# decompressed = huffman\_decompress(compressed, tree)

# print(f"\nDecompressed text: {decompressed}")

# print("Success!" if decompressed == text else "Error in decompression")