Name	Adwait Purao
UID no.	2021300101

Experiment 10	
AIM :	Perform Image Compression and Decompression
OBJECTIVE:	● To reduce data redundancy in files (images and documents) during compression. ● To conserve more hardware space and transmission bandwidth by reducing the file size through compression. ● To enable faster and efficient data transfer without loss of quality by maintaining the quality of the file after decompression. ● To perform compression and decompression of files without losing any data or compromising the quality of the original file.
INTRODUCTION:	This paper proposes using the Gzip algorithm for image and document compression, which combines Lz77 and Huffman techniques. Compression and decompression are vital in document management and communication systems. Image compression, particularly, is advantageous in digital image processing. The goal is to reduce redundancy in images and documents to efficiently store or transmit data. This approach conserves hardware space and bandwidth. In our proposed system, we achieve reduced data size without quality loss.

BLOCK DIAGRAM:

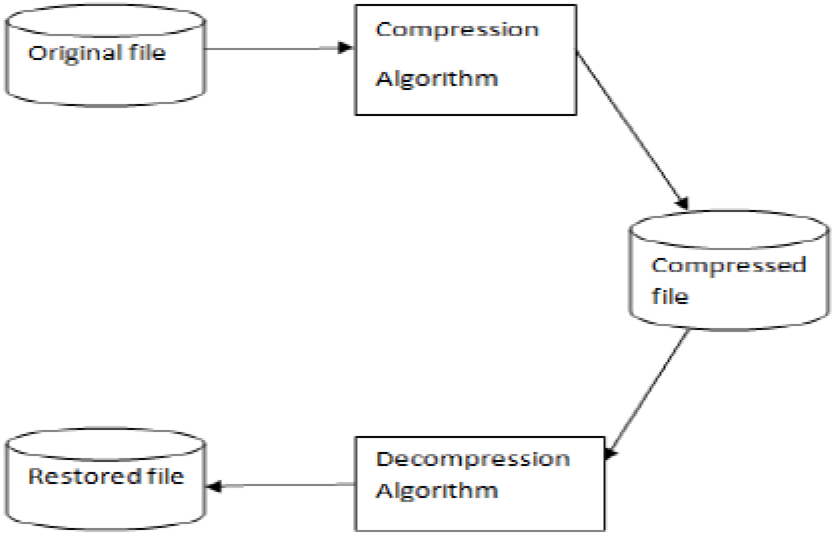


FIGURE 2. Process Flow

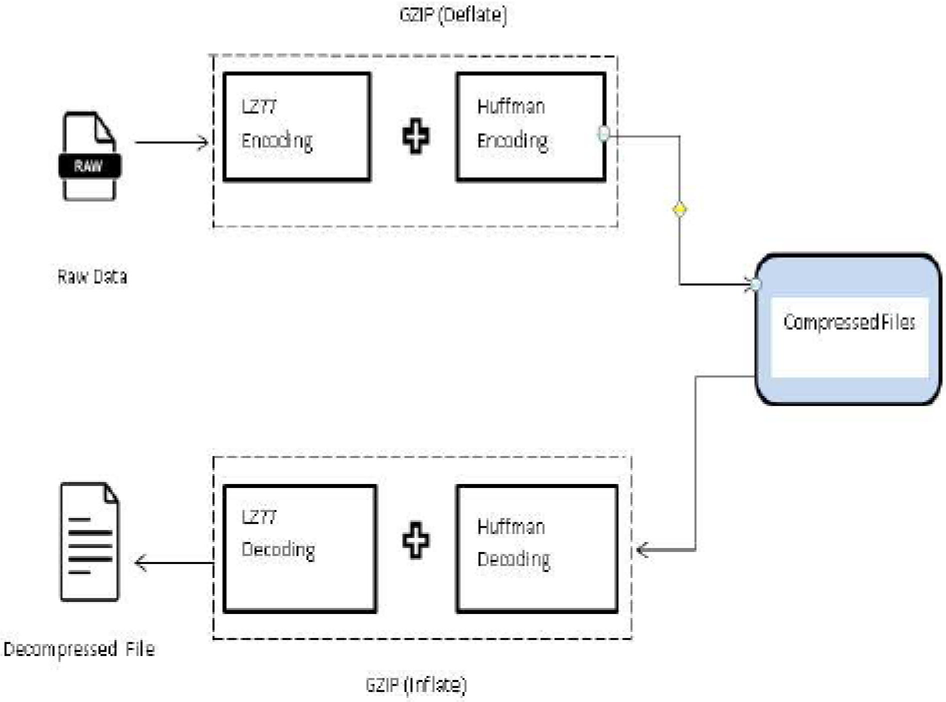
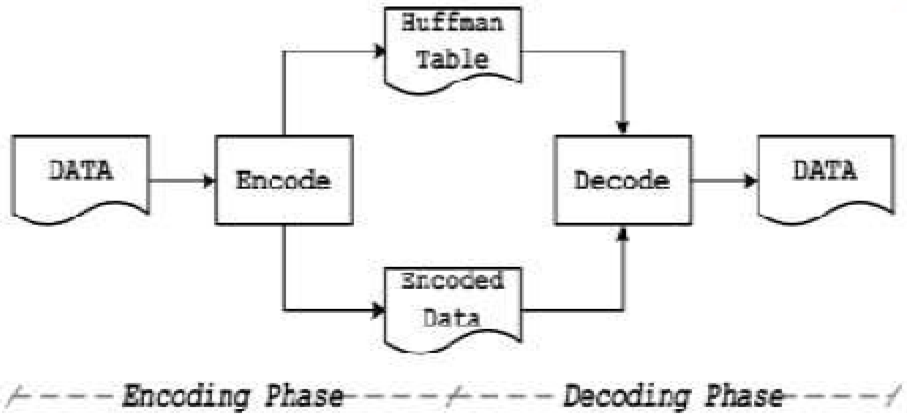
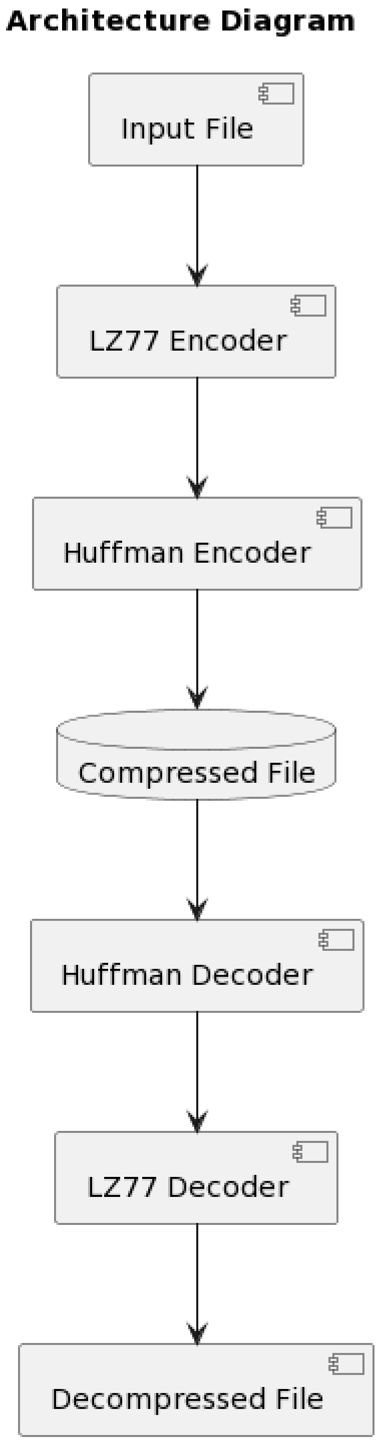


Figure 1. Architecture Diagram

	 Huffman Table DATA Encode Decode DATA Encoded Data Encoding Phase Decoding Phase FIGURE 5. Huffman block diagram



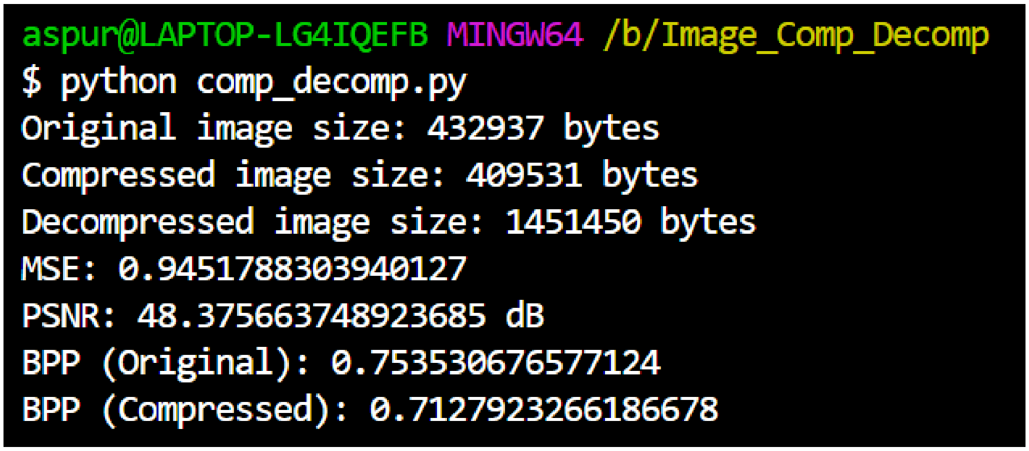
Block Diagram

IMPLEMENTATION:

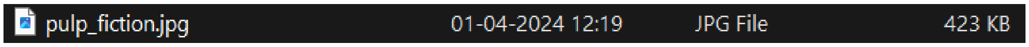
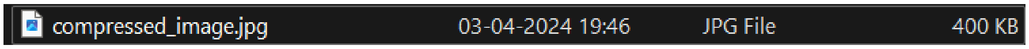
```python
from PIL import Image
import os
import numpy as np
import gzip

def compress_image(input_image_path, compressed_image_path, quality=80):
    original_image = Image.open(input_image_path)
    original_size = os.path.getsize(input_image_path)

    original_image.save(compressed_image_path, 'JPEG', quality=quality, optimize=True)
    compressed_size = os.path.getsize(compressed_image_path)

    return original_size, compressed_size

def decompress_image(compressed_image_path, decompressed_image_path):
    compressed_image = Image.open(compressed_image_path)
    compressed_image.save(decompressed_image_path, 'JPEG', quality=100)
    decompressed_size = os.path.getsize(decompressed_image_path)

    return decompressed_size

def calculate_metrics(original_image_path, compressed_image_path):
    original_image = Image.open(original_image_path).convert('RGB')
    compressed_image = Image.open(compressed_image_path).convert('RGB')

    original_image_array = np.array(original_image)
    compressed_image_array = np.array(compressed_image)

    mse = np.mean((original_image_array - compressed_image_array) ** 2)
    max_pixel_value = 255
    psnr = 20 * np.log10(max_pixel_value / np.sqrt(mse))

    original_image_size = os.path.getsize(original_image_path)
    compressed_image_size = os.path.getsize(compressed_image_path)
    bpp_original = (original_image_size * 8) / (original_image.size[0] * original_image.size[1])
    bpp_compressed = (compressed_image_size * 8) / (compressed_image.size[0] * compressed_image.size[1])

    return mse, psnr, bpp_original, bpp_compressed

# Example usage
input_image_path = 'B:\\Image_Comp_Decomp\\pulp_fiction.jpg'
compressed_image_path = 'compressed_image.jpg'
decompressed_image_path = 'decompressed_image.jpg'

original_size, compressed_size = compress_image(input_image_path, compressed_image_path)
decompressed_size = decompress_image(compressed_image_path, decompressed_image_path)

print(f'Original image size: {original_size} bytes')
print(f'Compressed image size: {compressed_size} bytes')
print(f'Decompressed image size: {decompressed_size} bytes')

mse, psnr, bpp_original, bpp_compressed = calculate_metrics(input_image_path, compressed_image_path)
print(f'MSE: {mse}')
print(f'PSNR: {psnr} dB')
print(f'BPP (Original): {bpp_original}')
print(f'BPP (Compressed): {bpp_compressed}')
```

OUTPUT:

Terminal:

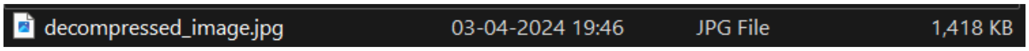


```
aspur@LAPTOP-LG4IQEFB MINGW64 /b/Image_Comp_Decomp
$ python comp_decomp.py
Original image size: 432937 bytes
Compressed image size: 409531 bytes
Decompressed image size: 1451450 bytes
MSE: 0.9451788303940127
PSNR: 48.375663748923685 dB
BPP (Original): 0.753530676577124
BPP (Compressed): 0.7127923266186678
```

	Input Image: **Size :**  pulp_fiction.jpg 01-04-2024 12:19 JPG File 423 KB **Compressed Image:** **Size :**  compressed_image.jpg 03-04-2024 19:46 JPG File 400 KB

<table>
<tr>
<td></td>
<td>

Decompressed Image:
Size :



decompressed_image.jpg 03-04-2024 19:46 JPG File 1,418 KB

</td>
</tr>
<tr>
<td>

REFERENCE:

</td>
<td>

K. Anand, M. Priyadharshini and K. Priyadharshini, "Compression And Decompression Of Files Without Loss Of Quality," 2023 International Conference on Networking and Communications (ICNWC), Chennai, India, 2023, pp. 1-6, doi: 10.1109/ICNWC57852.2023.10127236. keywords: {Image quality;Image coding;Communication systems;Digital images;Redundancy;Data compression;Bandwidth;Lz77;Gzip;Hybrid algorithm},

https://ieeexplore.ieee.org/document/10127236

</td>
</tr>
<tr>
<td colspan="2">

CONCLUSION:
The proposed Gzip algorithm, integrating LZ77 and Huffman coding, efficiently compresses and decompresses images and documents without sacrificing quality. It reduces data redundancy, conserves space, boosts bandwidth, and maintains file integrity, offering a practical solution for compression and decompression needs.

</td>
</tr>
</table>