資料結構與圖論演算法

Assignment 7

110704008 羅瑋翔

xianglo9121.mg10@nycu.edu.tw

Statement of the Problem

In this assignment, we are implementing Huffman coding. This is an idea for lossless data compression. The implementation involves reading a file first, recording the frequency of each character occurred, sorting the characters based on their frequencies in ascending order, and creating a new node by summing the frequencies of the two least frequent characters at the front of the sorted array. The newly created node replaces the two original characters, and pointers are used to link to these two elements.

This process is repeated until only the root node remains in the array that contains the nodes, resulting in the completion of the Huffman tree. After constructing the Huffman tree, we assign add 0 to the code if the pointer goes left and 1 if the pointer goes right. Following these steps, each character will have a distinct code and the length of the code might be different. Placing frequently occurring characters closer to the root reduces the code length of it. By this, we can minimize the usage of storage. And we save the context in the file with binary strings. Hence, we have the compressed file.

In the decoding process, we utilize the Huffman tree constructed during encoding. The file is read byte by byte, and for each bit (0 or 1), we determine whether to traverse to the left or right child node of the current node in the tree. Upon reaching a leaf node representing a character, that character is written to the file for decoding. By continuously doing this step, we will finally have the original context file.

Main Result

(a) First, I created a structure named "node," which includes a character to record the symbol represented by the node, an integer to track the frequency of the symbol, and two pointers to indicate the two child nodes of the current node. Next, I established an array to store nodes and defined the values for characters and their frequencies within the nodes. The characters include a to z, A to Z, and some commonly used punctuation marks, newline characters, and spaces, among others. All character frequencies are initialized to zero.

Subsequently, through file reading, I obtained the frequency of each character present in the file by initializing their occurrences to zero and updating the counts accordingly. After reading the file, I sorted the characters based on their frequencies in ascending order. In this case, I used the quicksort algorithm for sorting.

Next, I began constructing the Huffman tree using the method mentioned in the statement. After sorting the array, I added the frequencies of the two least frequent elements at the front of the array to create a new node, replacing the two elements. This process was repeated until there was only one node left in the array. During the tree construction, I encoded the characters by traversing the tree from the root to the left or right child nodes. The resulting codes were then stored in an array for the next steps in the process.

The next step involves compressing the file. Since we need to compare the sizes of the uncompressed and compressed files, I saved the compressed file separately. Initially, I created an empty string to store the binary encoding of the text. In this process, I used the get function to access each character in the file, converted it into its

corresponding Huffman code, and sequentially added the code to the empty string. After accessing all the characters in the file, I started grouping every 8 bits into a set (1 byte). If the last set does not contain 8 bits, it is directly stored.

Finally, for the decoding part, since the file being read consists of binary characters, it is necessary to specify the binary mode using ios::binary when opening the file. During file reading, I grouped every eight bits and converted the binary encoding back to text. I achieved this by utilizing the Huffman tree created earlier, starting from the root and traversing to the left or right child nodes based on encountering 1 or 0 in the encoding. If the pointer reaches null, it indicates reaching a leaf node, allowing me to access the character in the node and write it to the file. Following these steps, we can obtain the original text file.

(b) I utilized arrays, strings, and constructed a structure named "node" for my implementation. I used an array to store nodes, facilitating subsequent sorting and the construction of the Huffman tree. A string was employed to store the character encodings. The functionality of the "node" structure is like a map, containing key-value pairs, where the key represents a character, and the value represents the frequency of the character in the file. Additionally, the structure includes two pointers pointing to two child nodes, initialized to null. These pointers are frequently used in both compression and decompression processes. With this structure, it becomes convenient and efficient to quickly tally the occurrences of each character in the file.

(c) Program with Comments

```
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

struct Node
{
    char c;
    int freq;
    Node *left;
    Node *right;

    Node() : c('\0'), freq(0), left(nullptr), right(nullptr) {}

    Node(char c, int freq, Node *l = nullptr, Node *r = nullptr)
        : c(c), freq(freq), left(l), right(r) {}
};

void defNodes(Node nodes[]);

void recordFreq(Node nodes[], char c);

void sort(Node nodes[], int left, int right);

Node *buildHuffmanTree(Node nodes[], string encodingArray[]);

void generateHuffmanCodes(Node *root, string code, string
encodingArray[]);

void compressHuffman(ifstream &inFile, ofstream &outFile, Node *root,
string encodingArray[]);

void decodeHuffman(ifstream &inFile, ofstream &outFile, Node *root);

int main()
{
```

```
    ifstream inFile("text.txt");
    ofstream outFile("compress_ver.txt", ios::binary);
    int sum = 0;

    if (!inFile.is_open() || !outFile.is_open())
    {
        cout << "Error opening input file." << endl;
        return 1;
    }

    char c;
    Node *nodes = new Node[62];//the alphabet has 26 letters. 26*2 = 52
and there are 10 commonly used symbols.
    defNodes(nodes);//call this function to initialize the nodes array

    while (inFile.get(c))
        recordFreq(nodes, c);//calculate the frequency of each character

    // Create an array to store Huffman codes for each character
    string encodingArray[256] = {""};
    sort(nodes, 0, 61);

    Node *root = buildHuffmanTree(nodes, encodingArray);
    compressHuffman(inFile, outFile, root, encodingArray);
    inFile.close();
    outFile.close();

    inFile.open("compress_ver.txt", ios::binary);
    outFile.open("decompress_ver.txt");
    if (!inFile.is_open() || !outFile.is_open())
    {
        cout << "Error opening input file." << endl;
        return 1;
    }
    decodeHuffman(inFile, outFile, root);

    inFile.close();
    outFile.close();
```

```
    delete[] nodes;
    delete root;

    return 0;
}

void defNodes(Node nodes[])
{
    for (int i = 0; i < 26; i++)
    {
        nodes[i].c = 'a' + i;
        nodes[i].freq = 0;
    }
    for (int i = 0; i < 26; i++)
    {
        nodes[i + 26].c = 'A' + i;
        nodes[i + 26].freq = 0;
    }
    nodes[52].c = ' ';
    nodes[52].freq = 0;
    nodes[53].c = '.';
    nodes[53].freq = 0;
    nodes[54].c = ',';
    nodes[54].freq = 0;
    nodes[55].c = '(';
    nodes[55].freq = 0;
    nodes[56].c = ')';
    nodes[56].freq = 0;
    nodes[57].c = '"';
    nodes[57].freq = 0;
    nodes[58].c = '\'';
    nodes[58].freq = 0;
    nodes[59].c = '\n';
    nodes[59].freq = 0;
    nodes[60].c = '-';
    nodes[60].freq = 0;
    nodes[61].c = '?';
```

```
    nodes[61].freq = 0;
}

void recordFreq(Node nodes[], char c)
{
    if (c >= 'a' && c <= 'z')
        nodes[c - 'a'].freq++;
    else if (c >= 'A' && c <= 'Z')
        nodes[c - 'A' + 26].freq++;
    else if (c == ' ')
        nodes[52].freq++;
    else if (c == '.')
        nodes[53].freq++;
    else if (c == ',')
        nodes[54].freq++;
    else if (c == '(')
        nodes[55].freq++;
    else if (c == ')')
        nodes[56].freq++;
    else if (c == '"')
        nodes[57].freq++;
    else if (c == '\'')
        nodes[58].freq++;
    else if (c == '\n')
        nodes[59].freq++;
    else if (c == '-')
        nodes[60].freq++;
    else if (c == '?')
        nodes[61].freq++;
    else
        cout << "Error: " << c << " is not a valid character." << endl;
}

void sort(Node nodes[], int left, int right)
{
    int i = left, j = right + 1;
    if (left < right)
    {
```

```
        while (i < j)
        {
            while (nodes[++i].freq < nodes[left].freq)
                ;
            while (nodes[--j].freq > nodes[left].freq)
                ;
            if (i >= j)
                break;
            Node tmp = nodes[i];
            nodes[i] = nodes[j];
            nodes[j] = tmp;
        }
        Node tmp = nodes[j];
        nodes[j] = nodes[left];
        nodes[left] = tmp;
        sort(nodes, left, j - 1);
        sort(nodes, j + 1, right);
    }
}

Node *buildHuffmanTree(Node nodes[], string *encodingArray)
{
    int left = 0;
    int right = 61; // Assuming you have 62 nodes

    // Build the Huffman Tree using a bottom-up approach
    while (left < right)
    {
        Node *newNode = new Node('\0', nodes[left].freq + nodes[left +
1].freq, new Node(nodes[left]), new Node(nodes[left + 1]));
        nodes[left + 1] = *newNode;
        left++;
        sort(nodes, left, right);
    }

    // Generate Huffman codes starting from the root
    generateHuffmanCodes(&nodes[right], "", encodingArray);
```

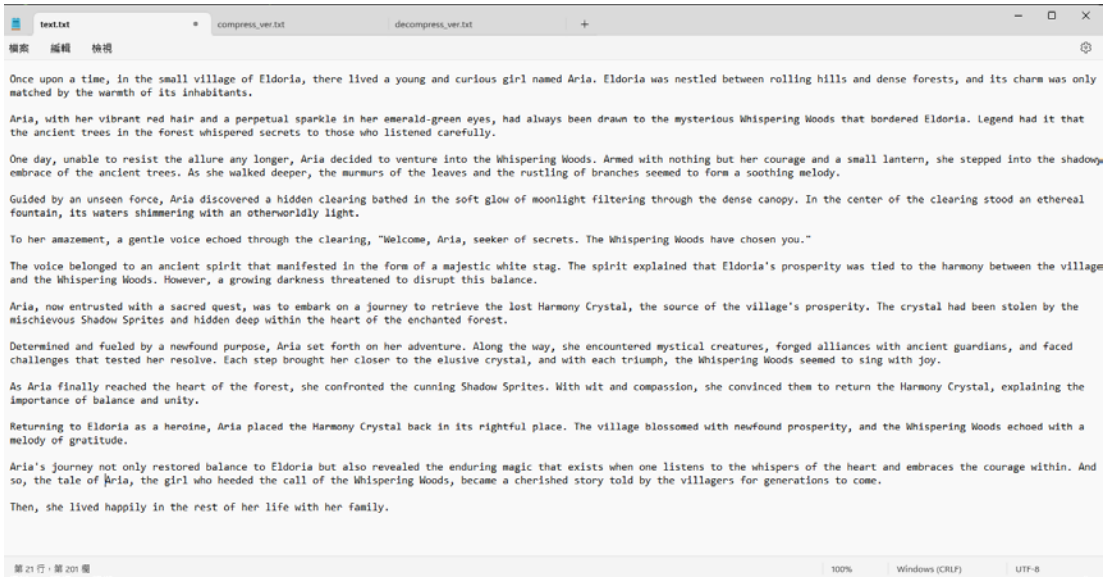
```
    return new Node(nodes[right]);
}

void generateHuffmanCodes(Node *root, string code, string
encodingArray[])
{
    if (root->left == nullptr && root->right == nullptr)
    {
        encodingArray[static_cast<unsigned char>(root->c)] = code; //
Store the mapping
        return;
    }

    // Recursively generate codes for the left and right branches
    generateHuffmanCodes(root->left, code + "0", encodingArray);
    generateHuffmanCodes(root->right, code + "1", encodingArray);
}

void compressHuffman(ifstream &inFile, ofstream &outFile, Node *root,
string encodingArray[])
{
    inFile.clear(); // by this we can read the file again
    inFile.seekg(0, ios::beg);

    string encodedText = "";
    char c;
    // Read the input file and append the Huffman code for each
character
    while (inFile.get(c))
        encodedText += encodingArray[static_cast<unsigned char>(c)];

    int sum = encodedText.length();
    // Variables to store the bits and count of bits in the buffer
    unsigned char buffer = 0;
    int bitCount = 0;

    for (size_t i = 0; i < encodedText.length(); ++i)
    {
```

```
        buffer = (buffer << 1) | (encodedText[i] - '0');// Append the
current bit to the buffer
        ++bitCount;
        // If the buffer is full (8 bits), write it to the output file
        if (bitCount == 8)
        {
            outFile.put(buffer);
            buffer = 0;
            bitCount = 0;
        }
    }
    // If there are remaining bits in the buffer, write them to the
output file
    if(bitCount > 0)
    {
        buffer = buffer << (8 - bitCount);
        outFile.put(buffer);
    }
}

void decodeHuffman(ifstream &inFile, ofstream &outFile, Node *root)
{
    Node *cur = root;
    unsigned char buffer;
    int bitCount = 0;

    while (inFile.read(reinterpret_cast<char *>(&buffer),
sizeof(buffer)))
    {
        // Iterate over each bit in the byte
        for (int i = 7; i >= 0; --i)
        {
            bool bit = (buffer >> i) & 1;
            if (bit)
                cur = cur->right;// Move to the right child if the bit
is 1
            else
```

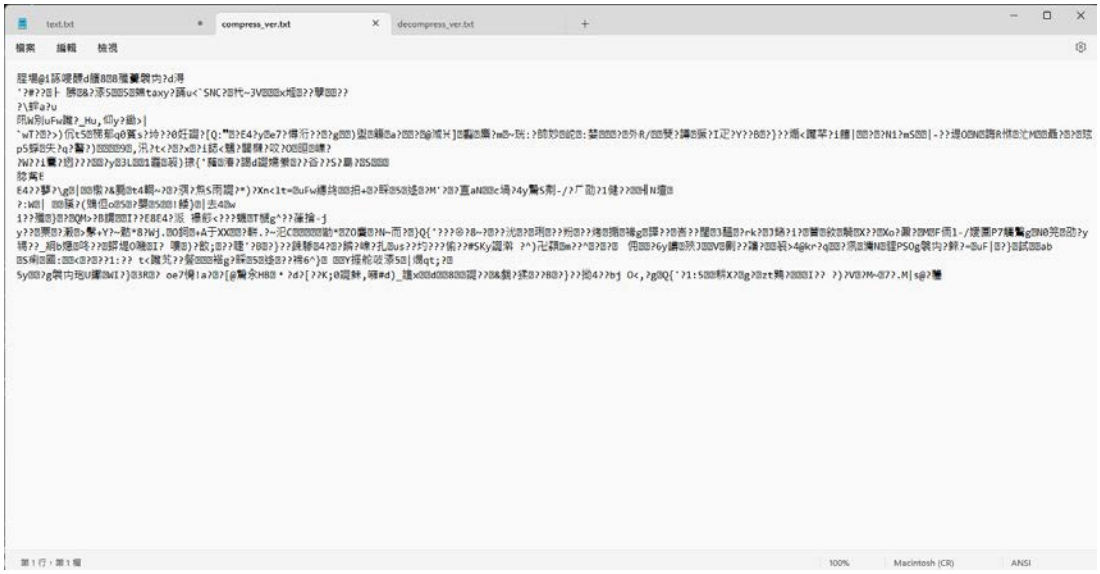
```
                cur = cur->left;// Move to the left child if the bit is
0
            // If a leaf node is reached, write the character to the
output file
            if (cur->left == nullptr && cur->right == nullptr)
            {
                outFile.put(cur->c);
                cur = root;
            }
        }
    }
}
```

(d) Outputs of the Compilation and the Execution of the Program

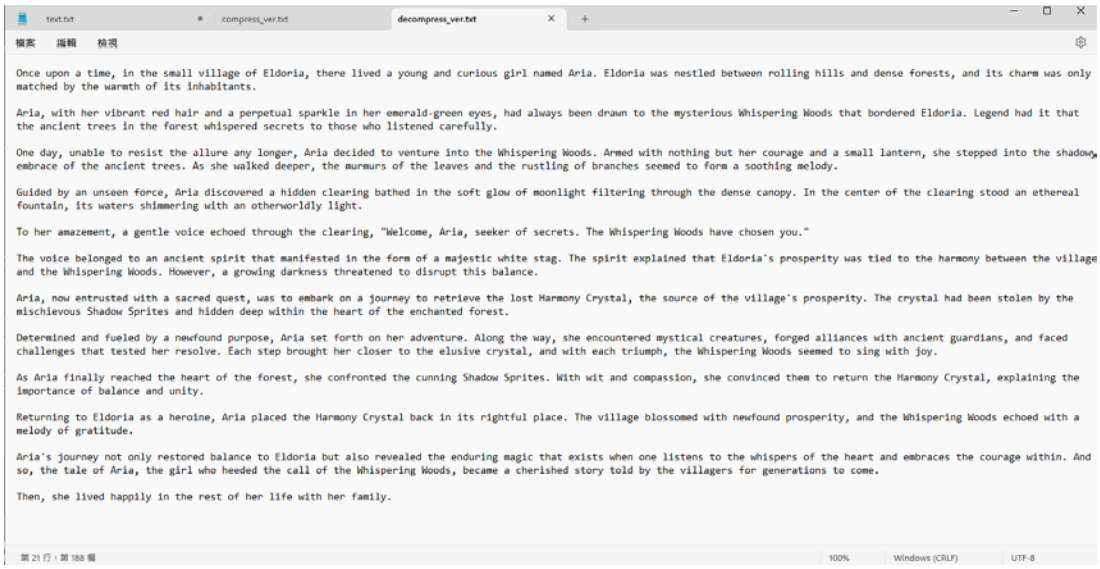
The input file (“text.txt”).



The compression file (“compression_ver.txt”).



The decompression file ("decompression_ver.txt”)



Once upon a time, in the small village of Eldoria, there lived a young and curious girl named Aria. Eldoria was nestled between rolling hills and dense forests, and its charm was only matched by the warmth of its inhabitants.

Aria, with her vibrant red hair and a perpetual sparkle in her emerald-green eyes, had always been drawn to the mysterious Whispering Woods that bordered Eldoria. Legend had it that the ancient trees in the forest whispered secrets to those who listened carefully.

One day, unable to resist the allure any longer, Aria decided to venture into the Whispering Woods. Armed with nothing but her courage and a small lantern, she stepped into the shadowy embrace of the ancient trees. As she walked deeper, the murmurs of the leaves and the rustling of branches seemed to form a soothing melody.

Guided by an unseen force, Aria discovered a hidden clearing bathed in the soft glow of moonlight filtering through the dense canopy. In the center of the clearing stood an ethereal fountain, its waters shimmering with an otherworldly light.

To her amazement, a gentle voice echoed through the clearing, "Welcome, Aria, seeker of secrets. The Whispering Woods have chosen you."

The voice belonged to an ancient spirit that manifested in the form of a majestic white stag. The spirit explained that Eldoria's prosperity was tied to the harmony between the village and the Whispering Woods. However, a growing darkness threatened to disrupt this balance.

Aria, now entrusted with a sacred quest, was to embark on a journey to retrieve the lost Harmony Crystal, the source of the village's prosperity. The crystal had been stolen by the mischievous Shadow Sprites and hidden deep within the heart of the enchanted forest.

Determined and fueled by a newfound purpose, Aria set forth on her adventure. Along the way, she encountered mystical creatures, forged alliances with ancient guardians, and faced challenges that tested her resolve. Each step brought her closer to the elusive crystal, and with each triumph, the Whispering Woods seemed to sing with joy.

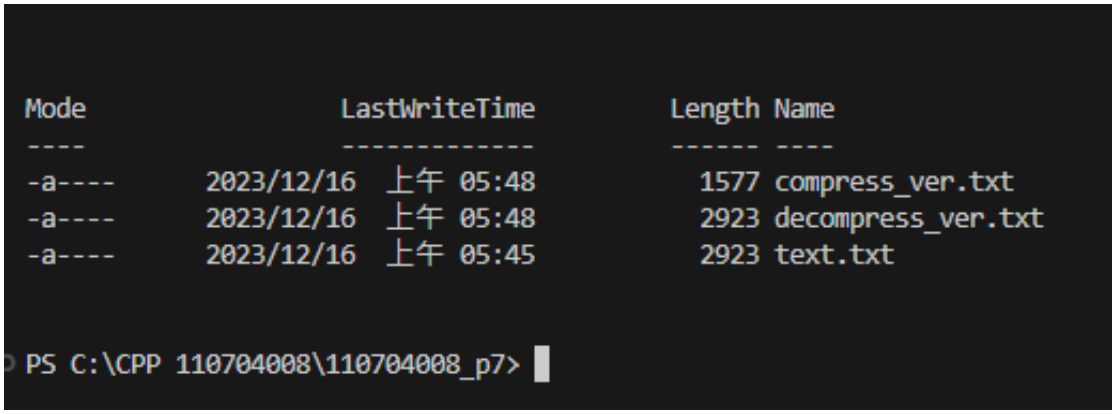
As Aria finally reached the heart of the forest, she confronted the cunning Shadow Sprites. With wit and compassion, she convinced them to return the Harmony Crystal, explaining the importance of balance and unity.

Returning to Eldoria as a heroine, Aria placed the Harmony Crystal back in its rightful place. The village blossomed with newfound prosperity, and the Whispering Woods echoed with a melody of gratitude.

Aria's journey not only restored balance to Eldoria but also revealed the enduring magic that exists when one listens to the whispers of the heart and embraces the courage within. And so, the tale of Aria, the girl who heeded the call of the Whispering Woods, became a cherished story told by the villagers for generations to come.

Then, she lived happily in the rest of her life with her family.

The storage comparing between 3 txt files.



Conclusions

This assignment involved implementing the Huffman tree, a lossless compression method. It took me quite a bit of time to work on this assignment, and even though the concept is simple (arranging and encoding characters from least to most frequent), I encountered some minor bugs in both the encoding and decoding functions, such as extra characters. In the decoding part, initially, I could only retrieve part of the text file. Later, I realized that I forgot to use ios::binary when reading the compressed file with infile (representing binary data). This was a critical oversight, and I'll remember it in the future. Despite the time invested, achieving both compression and decompression was rewarding, and I learned a lot during this assignment, identifying areas where my understanding was unclear.