《数据结构与算法》实验报告

实验名称	哈夫曼编/译码器				
姓名	叶鹏	学号	20020007095	日期	2022/4/22
实验内容	利用哈夫曼编码进行通信可以大大提高信道利用率，缩短信息传输时间，降低传输成本。但是，这要求在发送端通过一个编码系统对待传数据预先编码，在接收端将传来的数据进行译码（复原）。对于双工信道（即可以双向传输信息的信道），每端都需要一个完整的编/译码系统。试为这样的信息收发站写一个哈夫曼码的编/译码系统。				
实验目的	掌握哈夫曼树。				

实验步骤	哈夫曼编码是一种压缩技术，其压缩过程不会丢失细节，具体的编码过程可以分为以下步骤： 1. 以每一位字符的出现频率作为权值建立一棵哈夫曼树 2. 为每一位字符生成其对应编码 3. 以**相同的树**对一串编码进行解码 有了大体步骤，我们可以一步一步完成本次实验 1) 初始化(Initialization) 考虑建立一个菜单界面，以便输入关键字执行指定功能

```
53      void Menu() {
54          cout << "------------------|| huffmanCode Program ||------------------" << endl;
55          cout << endl;
56          cout << "                      1.Initialization" << endl;
57          cout << "                      2.Encoding" << endl;
58          cout << "                      3.Decoding" << endl;
59          cout << "                      4.Print" << endl;
60          cout << "                      5.Tree printing" << endl;
61          cout << "                      0.Exit" << endl;
62          cout << endl;
63          cout << "------------------|| 20020007095 YePeng  ||------------------" << endl;
64
65          cout << "Your choice: ";
66      }
```

```
C:\Users\ASUS\Documents\DataStructure\laboratory\chapter_9-11\lab9-11\x64\Debug\lab9-11.exe
------------------|| huffmanCode Program ||------------------

                      1.Initialization
                      2.Encoding
                      3.Decoding
                      4.Print
                      5.Tree printing
                      0.Exit

------------------|| 20020007095 YePeng  ||------------------
Your choice:
```

从终端读入字符集大小 n，以及 n 个字符和 n 个权值，建立哈夫曼树，并将它存入文件 hfmTree 中。

```cpp
// 从终端读入字符集大小n，以及n个字符和n个权值，建立哈夫曼树，并将它存入文件hfmTree中。
void Solution::Initialization()
{
    int n;// size
    cout << "the size of input: ";
    cin >> n;
    arr.resize(n);
    weight.resize(n);

    cout << "the characters: ";
    for (int i = 0; i < n; ++i)cin >> arr[i];
    cout << "the weights: ";
    for (int i = 0; i < n; ++i)cin >> weight[i];

    // store data in hfmTree.txt
    fstream file;
    file.open("hfmTree.txt", ios::in | ios::out | ios::trunc);

    for (int i = 0; i < n; ++i)file << arr[i] << " ";
    file << endl;
    for (int i = 0; i < n; ++i)file << weight[i] << " ";

    file.close();
```

但是这么写太麻烦了，我决定使用一个函数来统计 ToBeTran 文件中每个字符出现的次数作为其权值，直接存入内存中，这样免去了 debug 过程中每次手动输入的麻烦，考虑使用**哈希表**来储存每个字符以及其出现的次数，这样可以实现常数时间内的查找

```cpp
void Solution::calcAllNeeded()
{
    // read from ToBeTran
    fstream file;
    file.open("ToBeTran.txt", ios::in);
    string text;
    file >> text;

    // count each character
    for (int i = 0; i < text.size(); ++i)freq[text[i]]++;
}
```

建立哈夫曼树，因为建立过程需要每次都选择权重最低的两个结点，势必每次操作都需要排序，我们可以使用**堆**的数据结构，因为**堆本身有序**，每次插入或删除其中元素都不改变其有序性，所以用来建立哈夫曼树再适合不过，因此我们可以先声明树结点的结构

```cpp
// huffman tree node
struct TreeNode
{
    char val;// the data of the node
    int weight;

    TreeNode* left, * right;

    TreeNode(char _val, int _weight) {
        this->val = _val;
        this->weight = _weight;
        left = right = NULL;
    }
};
```

考虑到 priority_queue 的语法，对自定义数据结构的排序需要一个自定义结构来实现

```cpp
// struct for compare tree node
struct Compare
{
    bool operator()(TreeNode* left, TreeNode* right) {
        return left->weight < right->weight;
    }
};
```

在 cpp 中使用**优先队列**来实现堆结构

```cpp
void Solution::BuildHuffmanTree()
{
    int n = arr.size();// get the size

    priority_queue<TreeNode*, vector<TreeNode*>, Compare> nodes;
```

建立哈夫曼树的操作是，每次取堆中权重最小的两个结点，以他们的权重之和新建结点，原来的一左一右两个结点从堆中弹出，变为新结点的左右子节点，因为新节点不作为最后实际的解码输出结果，因此我们**用一个特殊符号’$’来标记**，直到堆中只剩一个结点，表示我们已经建立好了一棵哈夫曼树。

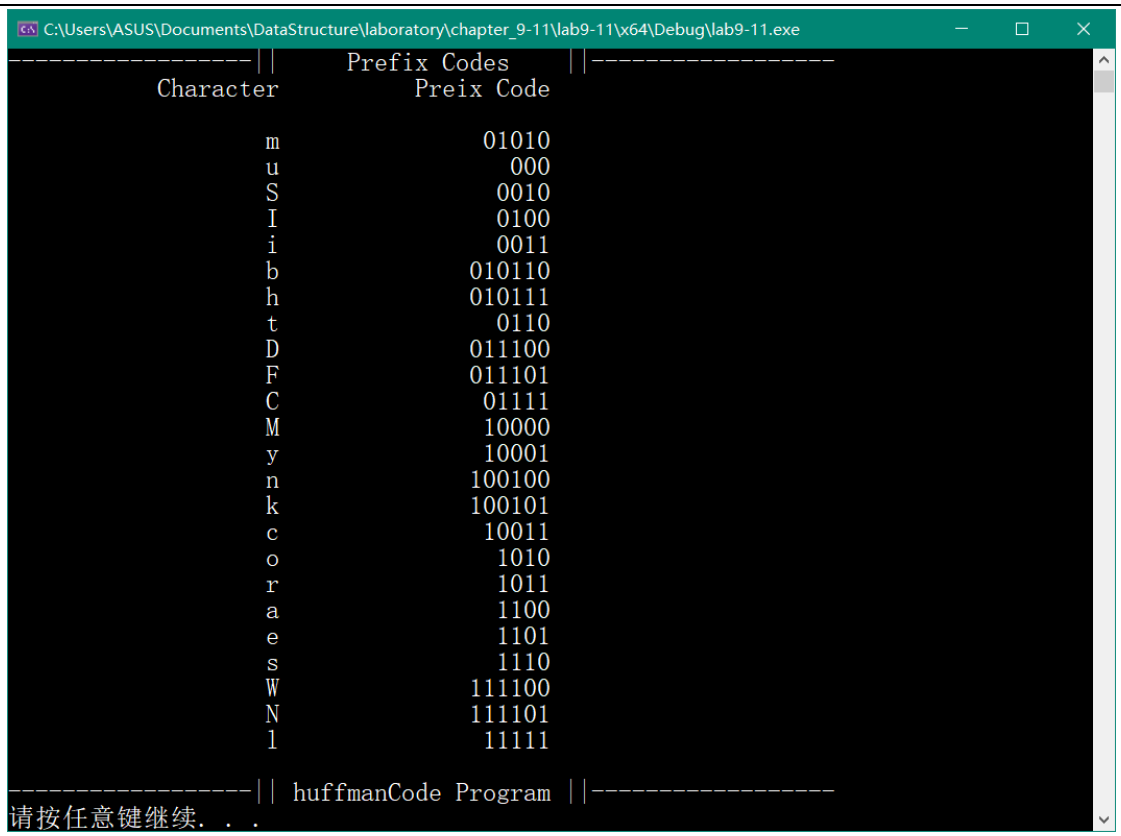
```
    priority_queue<TreeNode*, vector<TreeNode*>, Compare> nodes;

    for (int i = 0; i < n; ++i)nodes.emplace(new TreeNode(arr[i], weight[i]));

    while (nodes.size() != 1) {
        auto left = nodes.top();
        nodes.pop();

        auto right = nodes.top();
        nodes.pop();

        auto newNode = new TreeNode('$', left->weight + right->weight);
        newNode->left = left;
        newNode->right = right;

        nodes.emplace(newNode);
    }
}
```

通过哈夫曼树计算每个字符的前缀码，储存在哈希表当中

```
void Solution::storeCodes(TreeNode* node, string prefix)
{
    if (!node)return;
    if (node->val != '$')
        prefixCodes[node->val] = prefix;
    storeCodes(node->left, prefix + '0');
    storeCodes(node->right, prefix + '1');
}
```

测试将 TobeTran 文件中的内容进行 Initialization

```
PowerShell
PS C:\Users\ASUS\Documents\DataStructure\laboratory\chapter_9-11\lab9-11\lab9-1
1> type .\ToBeTran.txt
MyNameIsNoobMyDataStructureCourseSucksIamSureIWillFailInthisCourse
PS C:\Users\ASUS\Documents\DataStructure\laboratory\chapter_9-11\lab9-11\lab9-1
1>
```



2) 编码(Encoding)

先检查内存中是否存在哈夫曼树，如果没有，则从文件中读取，建立新的哈夫曼树

```
143 void Solution::Encoding()
144 {
145     // check if tree exist
146     if (root == NULL) {
147         // read data from hfmTree.txt
148         int n;// size
149         fstream file;
150         file.open("hfmTree.txt", ios::in);
151 
152         file >> n;
153         arr.resize(n);
154         weight.resize(n);
155 
156         for (int i = 0; i < n; ++i)file >> arr[i];
157         for (int i = 0; i < n; ++i)file >> weight[i];
158 
159         file.close();
160 
161         root = BuildHuffmanTree();
162     }
```

从 ToBrTran 中读取正文，通过查找哈希表对其编码，将编码结果储存在 CodeFile 中

```
    file.open("ToBeTran.txt", ios::in);

    string text;
    file >> text;

    file.close();

    file.open("CodeFile.txt", ios::in | ios::out | ios::trunc);

    for (int i = 0; i < text.size(); ++i) {
        file << prefixCodes[text[i]];
    }

    file.close();

    cout << "------------------||   Encoding Process  ||------------------" << endl;

    cout << setw(40) << right << "Successfully encoded!" << endl;

    cout << "------------------|| huffmanCode Program ||------------------" << endl;

    system("pause");
}
```

```
C:\Users\ASUS\Documents\DataStructure\laboratory\chapter_9-11\lab9-11\x64\Debug\lab9-11.exe
------------------||   Encoding Process  ||------------------
                  Successfully encoded!
------------------|| huffmanCode Program ||------------------
请按任意键继续. . .
```

编码结果：

```
PowerShell
PS C:\Users\ASUS\Documents\DataStructure\laboratory\chapter_9-11\lab9-11\lab9-1
1> type .\ToBeTran.txt
MyNameIsNoobMyDataStructureCourseSucksIamSureIWillFailInthisCourse
PS C:\Users\ASUS\Documents\DataStructure\laboratory\chapter_9-11\lab9-11\lab9-1
1> type .\CodeFile.txt
10000100011111011100010101101010011101111011010101001011010000100010111001100011011000010011010110001001101100001011110101111101000010111110110100100001001110010111100100110001010001000010111101010011110000111111111111011101110000111111101001001000110010111001111100111110100001011111101
PS C:\Users\ASUS\Documents\DataStructure\laboratory\chapter_9-11\lab9-11\lab9-1
1>
```

3) 译码(Decoding)

译码过程相当于遍历一次哈夫曼树，遇到 0 走向左子树，1 走向右子树，直到遍历到叶子节点，输出其字符，再重新遍历哈夫曼树，直到译出所有编码为止，将结果写入 Text File 文件

```
    TreeNode* cur = root;
    for (int i = 0; i < code.size(); ++i) {
        if (code[i] == '0')cur = cur->left;
        else cur = cur->right;

        // if leaf
        if (!cur->left && !cur->right) {
            file << cur->val;
            cur = root;
        }
    }

    file.close();

    cout << "------------------||   Decoding Process  ||------------------" << endl;

    cout << setw(40) << right << "Successfully decoded!" << endl;

    cout << "------------------|| huffmanCode Program ||------------------" << endl;

    system("pause");
}
```

```
C:\Users\ASUS\Documents\DataStructure\laboratory\chapter_9-11\lab9-11\x64\Debug\lab9-11.exe
------------------||   Decoding Process  ||------------------
                    Successfully decoded!
------------------|| huffmanCode Program ||------------------
请按任意键继续. . .
```

译码结果：

```
PowerShell
PS C:\Users\ASUS\Documents\DataStructure\laboratory\chapter_9-11\lab9-11\lab9-1
1> type .\TextFile.txt
MyNameIsNoobMyDataStructureCourseSucksIamSureIWillFailInthisCourse
PS C:\Users\ASUS\Documents\DataStructure\laboratory\chapter_9-11\lab9-11\lab9-1
1>
```

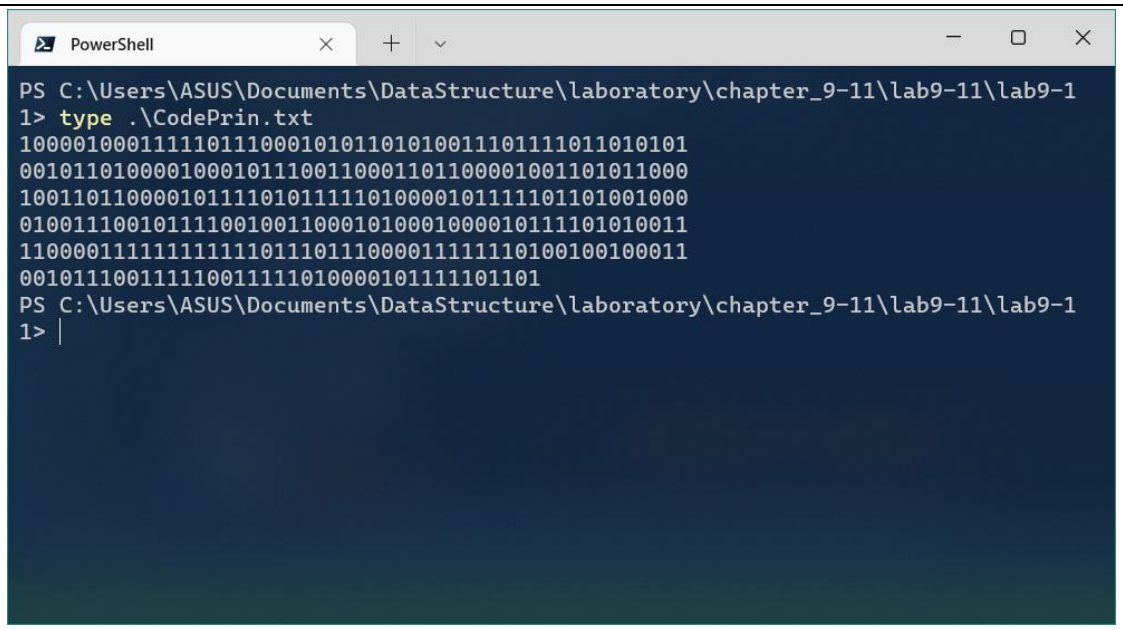
4) 印代码文件（Print）

将文件 CodeFile 以紧凑格式显示在终端上，每行 50 个代码。同时将此字符形式的编码文件写入文件 CodePrin 中

```cpp
void Solution::Print()
{
    // read from CodeFile
    fstream file;
    file.open("CodeFile.txt", ios::in);
    string code;
    file >> code;
    file.close();

    file.open("CodePrin.txt", ios::in | ios::out | ios::trunc);

    cout << "-------------||       CodeFile      ||-------------" << endl;
    for (int i = 0; i < code.size(); ++i) {
        if (i % 50 == 0) {
            cout << endl;
            file << endl;
        }
        cout << code[i];
        file << code[i];
    }
    cout << endl;
    cout << endl;
    cout << "-------------|| huffmanCode Program ||-------------" << endl;
    system("pause");
}
```

```
C:\Users\ASUS\Documents\DataStructure\laboratory\chapter_9-11\lab9-11\x64\Debug\lab9-11.exe
-------------||       CodeFile      ||-------------

10000100011111011100010101101010011101111011010101
00101101000010001011100110001101100001001101011000
10011011000010111101011111010000101111101101001000
01001110010111100100110001010001000010111101010011
11000011111111111101110111000011111110100100100011
00101110011111001111101000010111110110

-------------|| huffmanCode Program ||-------------
请按任意键继续. . .
```

CodePrin 文件：



5) 印哈夫曼树(Tree printing)

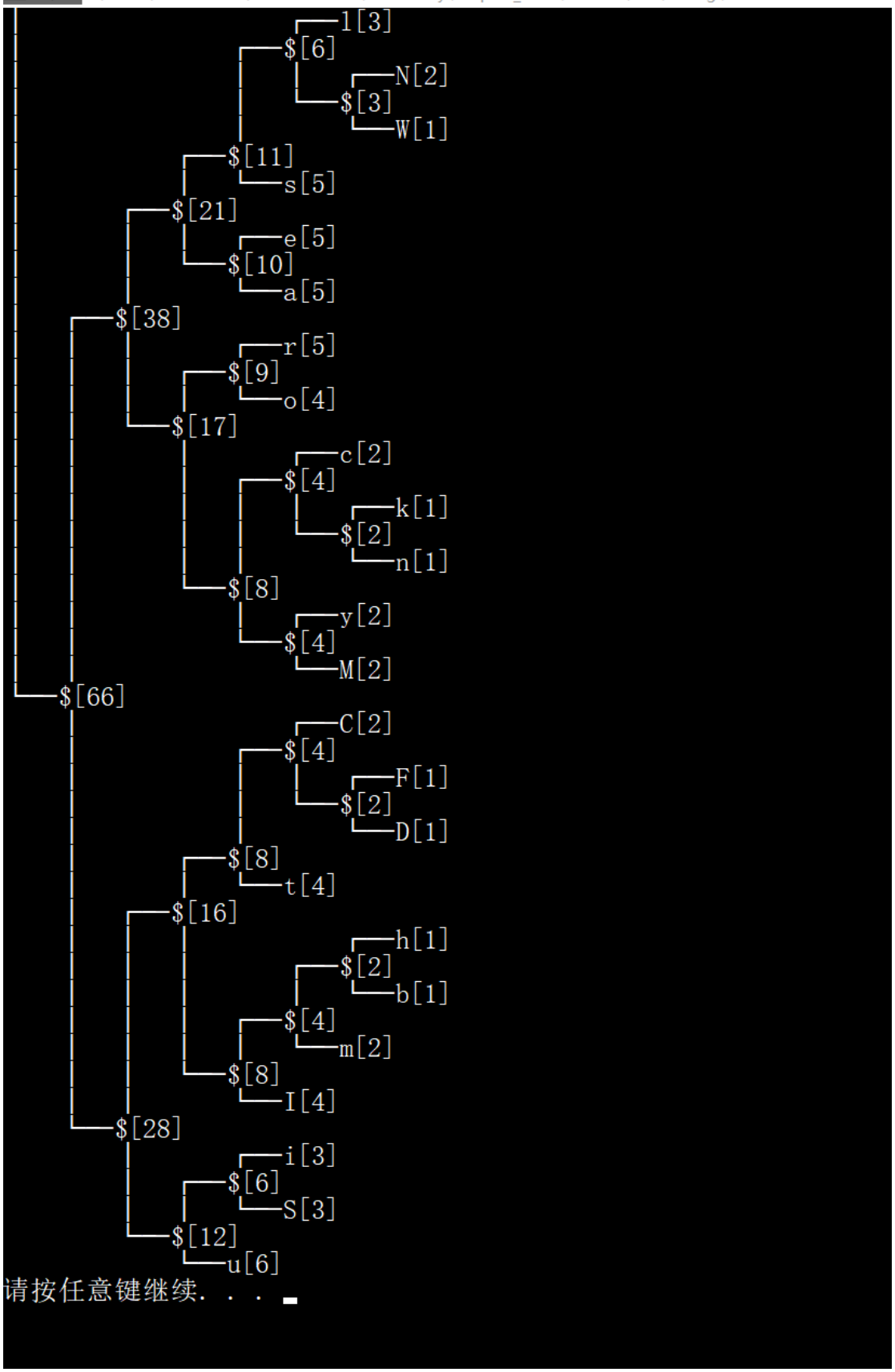
通过递归函数逐行打印哈夫曼树，越右的结点越在初始行，打印树的结点以及权值

```
388 void Solution::prettyPrintTree(TreeNode* node, vector<string>& tree, string prefix, bool i
389 {
390     if (node == nullptr) {
391         cout << "Empty tree";
392         return;
393     }
394
395     if (node->right) {
396         prettyPrintTree(node->right, tree, prefix + (isLeft ? "|   " : "    "), false);
397     }
398
399     cout << prefix + (isLeft ? "└── " : "┌── ") + node->val + '[' + to_string(node->weight
400     tree.emplace_back(prefix + (isLeft ? "└── " : "┌── ") + node->val + '[' + to_string(no
401
402
403     if (node->left) {
404         prettyPrintTree(node->left, tree, prefix + (isLeft ? "    " : "|   "), true);
405     }
406 }
407
```

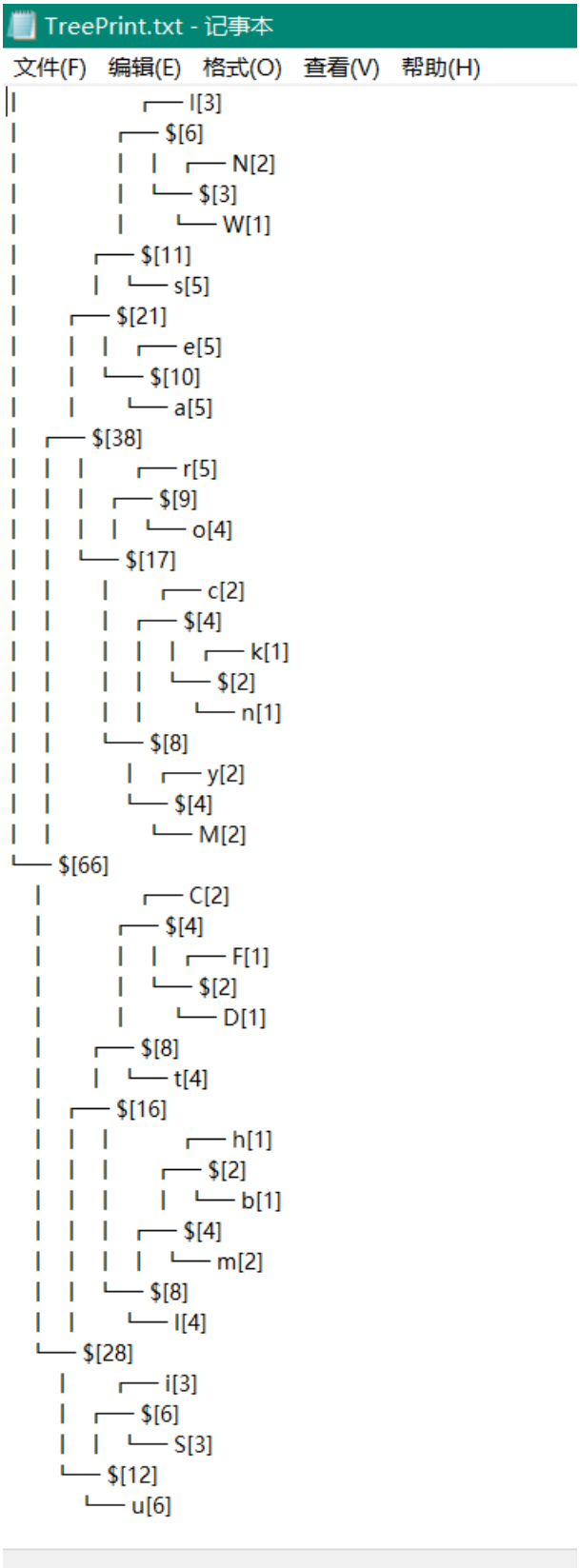
同时将此字符形式的哈夫曼树写入文件 TreePrint 中

打印树：

824 × 915 rs\ASUS\Documents\DataStructure\laboratory\chapter_9-11\lab9-11\x64\Debug\lab9-11.exe



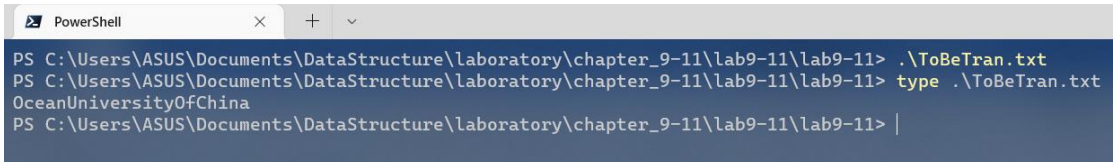
TreePrint 文件：


TreePrint.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
l[3]
$[6]
N[2]
$[3]
W[1]
$[11]
s[5]
$[21]
e[5]
$[10]
a[5]
$[38]
r[5]
$[9]
o[4]
$[17]
c[2]
$[4]
k[1]
$[2]
n[1]
$[8]
y[2]
$[4]
M[2]
$[66]
C[2]
$[4]
F[1]
$[2]
D[1]
$[8]
t[4]
$[16]
h[1]
$[2]
b[1]
$[4]
m[2]
$[8]
l[4]
$[28]
i[3]
$[6]
S[3]
$[12]
u[6]

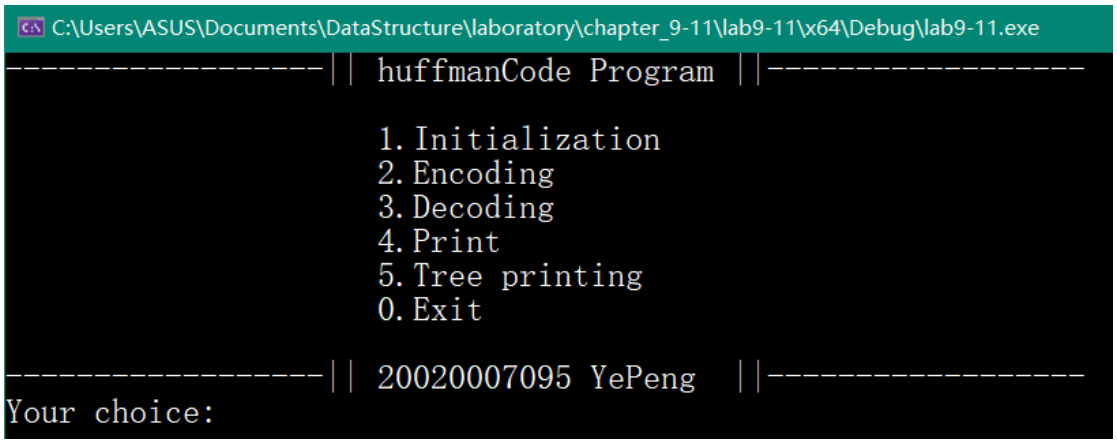
实验步骤

6) 测试一个不同的样例

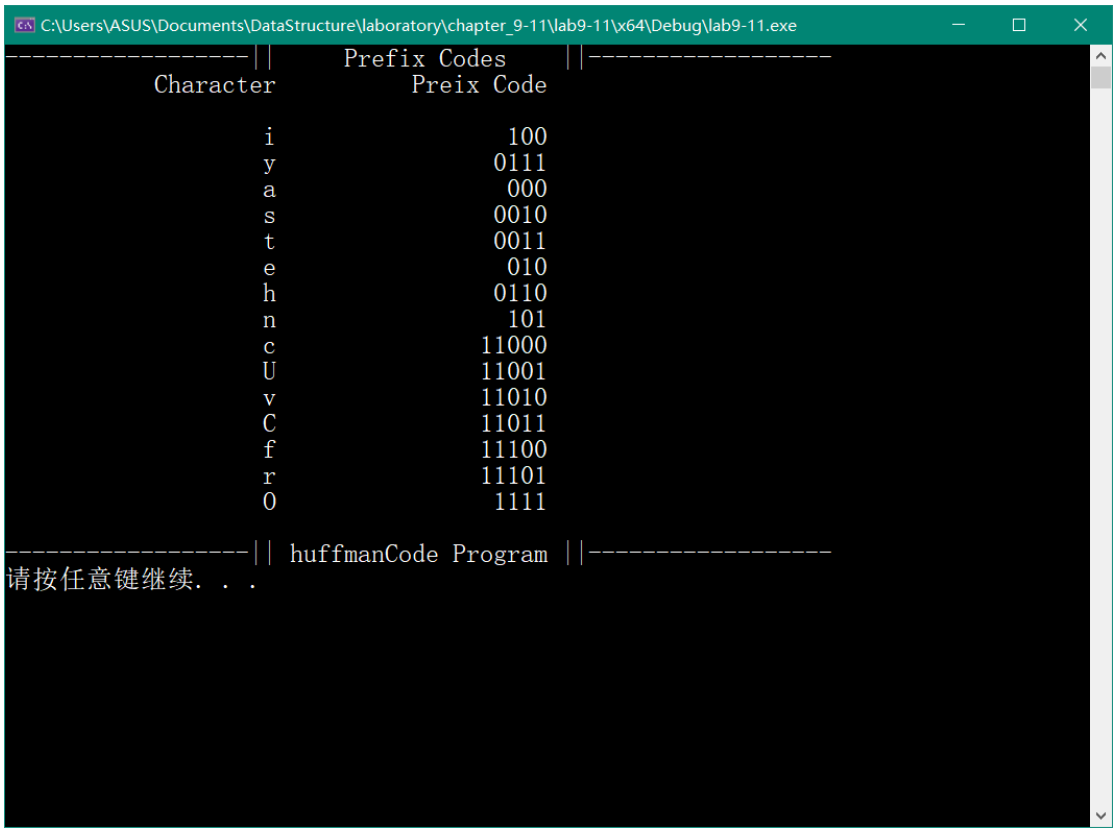
ToBeTran 文件：



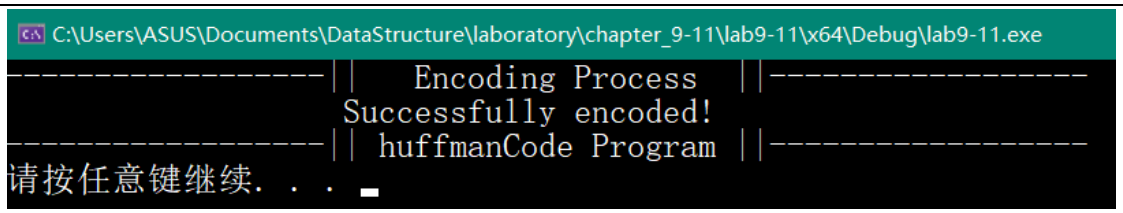
Menu：

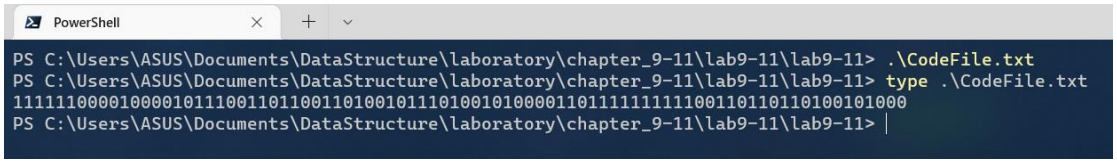


Initialization：

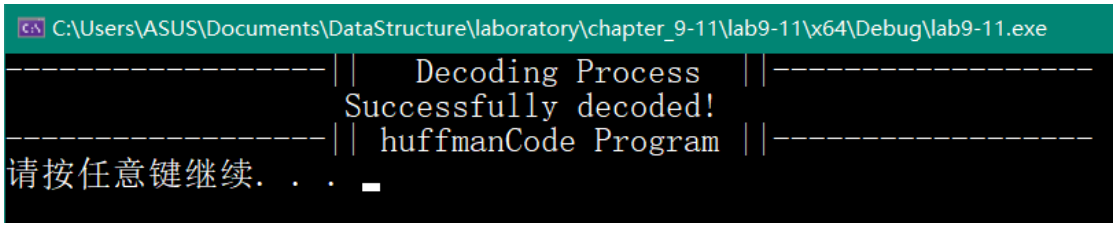


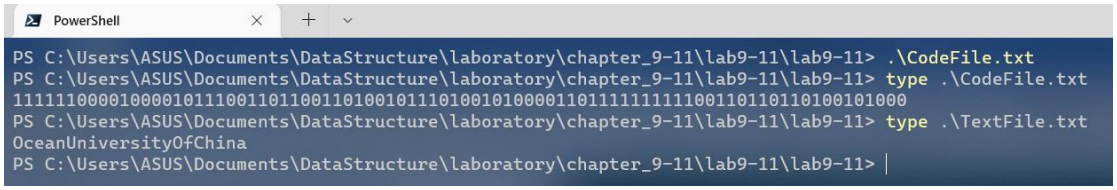
Encoding：



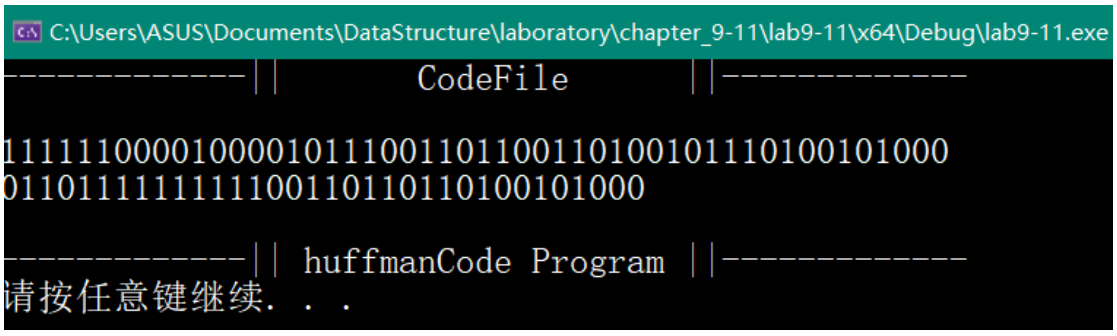


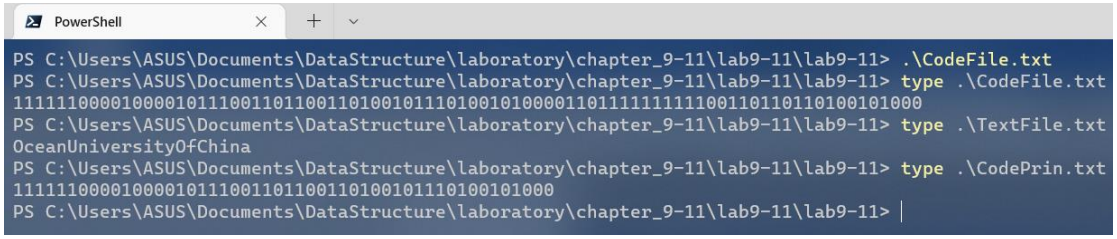
Decoding:



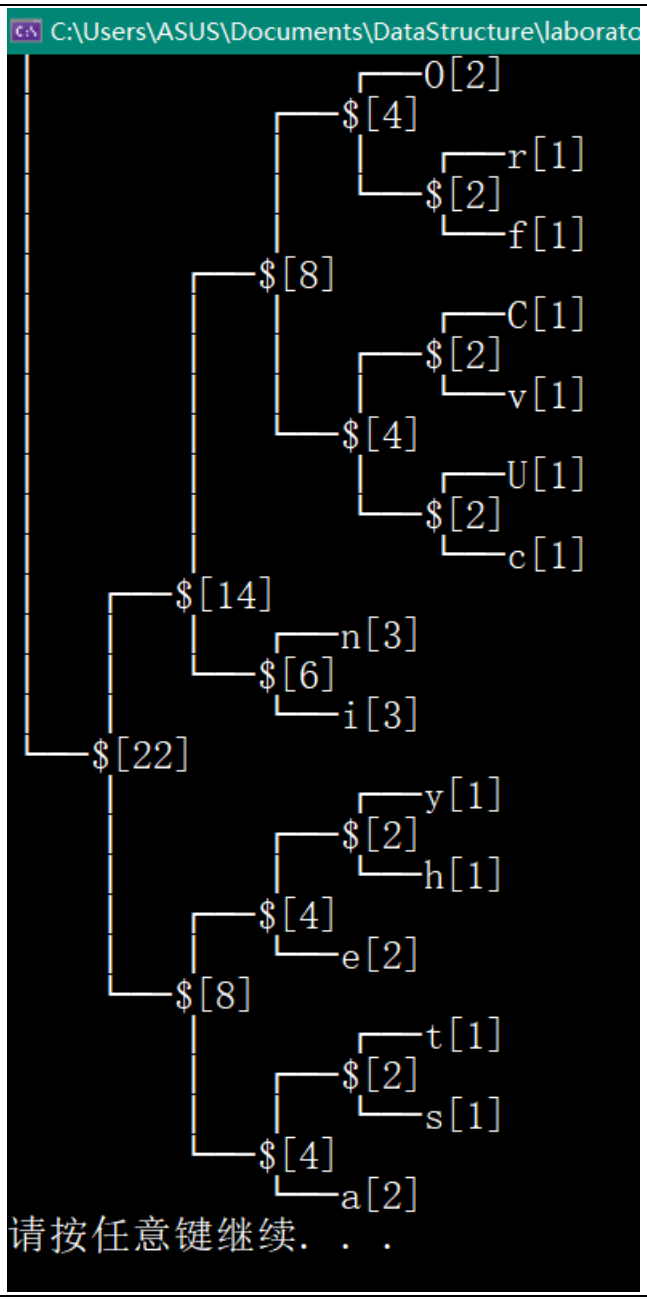


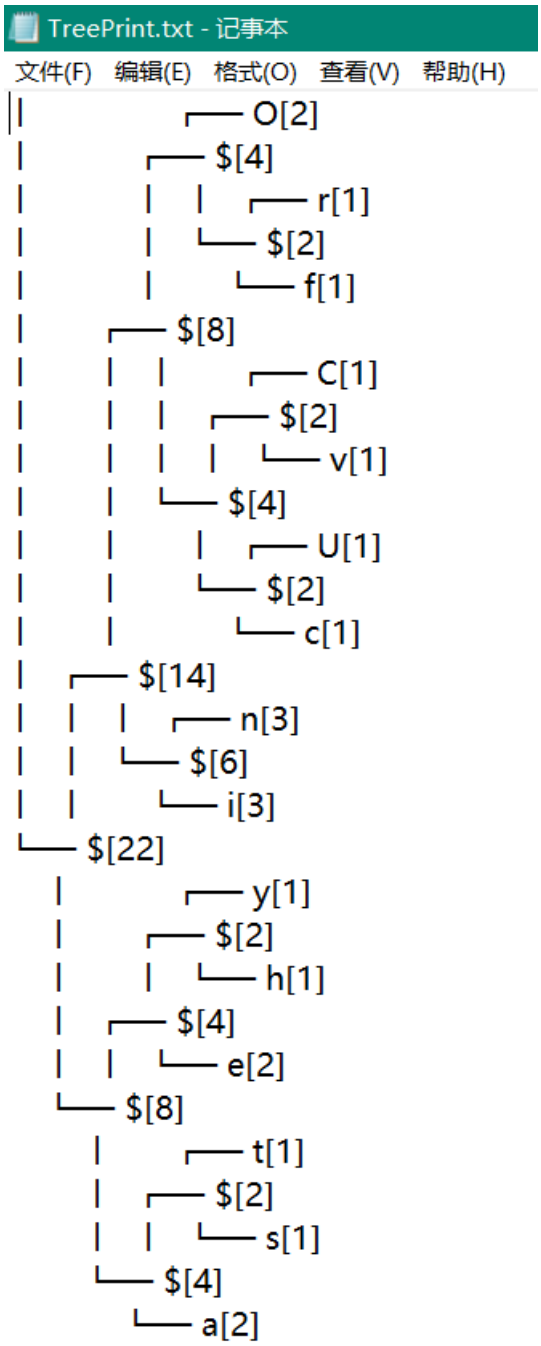
Print:





Tree Print:


C:\Users\ASUS\Documents\DataStructure\laborato
0[2]
$[4]
r[1]
$[2]
f[1]
$[8]
C[1]
$[2]
v[1]
$[4]
U[1]
$[2]
c[1]
$[14]
n[3]
$[6]
i[3]
$[22]
y[1]
$[2]
h[1]
$[4]
e[2]
$[8]
t[1]
$[2]
s[1]
$[4]
a[2]
请按任意键继续. . .



7） 源代码

```
#include <iostream>
#include <fstream>
#include <queue>
#include <vector>
#include <string>
#include <cstdlib>
#include <unordered_map>
```

```cpp
#include <iomanip>
using namespace std;

// huffman tree node
struct TreeNode
{
    char val;// the data of the node
    int weight;

    TreeNode* left, * right;

    TreeNode(char _val, int _weight) {
        this->val = _val;
        this->weight = _weight;
        left = right = NULL;
    }
};

// struct for compare tree node
struct Compare
{
    bool operator()(TreeNode* left, TreeNode* right) {
        return (left->weight > right->weight);
    }
};

class Solution {
private:
    int n;// size
    vector<char> arr;
    vector<int> weight;

    unordered_map<char, int> freq;// to store the frequency of character of the input data
    unordered_map<char, string> prefixCodes;// each character's prefix code

    TreeNode* root;// huffman tree

    vector<string> tree;// tree in graph

public:
    Solution() {
        root = NULL;
        n = 0;
    }
```

```
    void Menu() {
        cout << "------------------|| huffmanCode Program ||------------------" << endl;
        cout << endl;
        cout << "                   1.Initialization" << endl;
        cout << "                   2.Encoding" << endl;
        cout << "                   3.Decoding" << endl;
        cout << "                   4.Print" << endl;
        cout << "                   5.Tree printing" << endl;
        cout << "                   0.Exit" << endl;
        cout << endl;
        cout << "------------------|| 20020007095 YePeng  ||------------------" << endl;

        cout << "Your choice: ";
    }

    // 5 main functions
    void Initialization();
    void Encoding();
    void Decoding();
    void Print();
    void TreePrinting();

    TreeNode* BuildHuffmanTree();// build the tree
    void calcAllNeeded();// calulate num of character and the frequence of character
    void storeCodes(TreeNode* node, string prefix);// store each character's prefix code
    void prettyPrintTree(TreeNode* node, vector<string>& tree, string prefix = "", bool isLeft =
true);// Print tree
};

int main() {
    Solution solution;
    int choice;
    bool flag = true;

    while (flag) {
        system("cls");
        solution.Menu();
        cin >> choice;
        switch (choice)
        {
        case 1:
            system("cls");
            solution.Initialization();
```

```
            break;
        case 2:
            system("cls");
            solution.Encoding();
            break;
        case 3:
            system("cls");
            solution.Decoding();
            break;
        case 4:
            system("cls");
            solution.Print();
            break;
        case 5:
            system("cls");
            solution.TreePrinting();
            break;
        case 0:
            flag = false;
            break;
        default:
            system("cls");
            cout << "Error: choice not valid";
            system("timeout -t 5");
            system("cls");
            break;
        }
    }

    return 0;
}

// 从终端读入字符集大小n，以及n个字符和n个权值，建立哈夫曼树，并将它存入文件hfmTree中。
void Solution::Initialization()
{
    // Manual input

    //cout << "the size of input: ";
    //cin >> n;
    //arr.resize(n);
    //weight.resize(n);

    //cout << "the characters: ";
    //for (int i = 0; i < n; ++i)cin >> arr[i];
```

```
    //cout << "the weights: ";
    //for (int i = 0; i < n; ++i)cin >> weight[i];

    // store data in hfmTree.txt

    // automatic input
    calcAllNeeded();
    n = freq.size();
    arr.resize(n);
    weight.resize(n);

    int cnt = 0;
    for (auto it = freq.begin(); it != freq.end(); ++it) {
        arr[cnt] = (*it).first;
        weight[cnt] = (*it).second;
        cnt++;
    }

    fstream file;
    file.open("hfmTree.txt", ios::in | ios::out | ios::trunc);

    file << n;
    file << endl;
    for (int i = 0; i < n; ++i)file << arr[i] << " ";
    file << endl;
    for (int i = 0; i < n; ++i)file << weight[i] << " ";

    file.close();

    // build the tree
    root = BuildHuffmanTree();

    cout << "------------------||    Prefix Codes    ||------------------" << endl;
    cout << setw(20) << right << "Character" << setw(20) << right << "Preix Code" << endl;
    cout << endl;
    for (auto it = prefixCodes.begin(); it != prefixCodes.end(); ++it) {
        cout << setw(20) << right << (*it).first << setw(20) << right << (*it).second << endl;
    }
    cout << endl;
    cout << "------------------|| huffmanCode Program ||------------------" << endl;

    system("pause");
}
```

```
// 利用已建好的哈夫曼树（如不在内存，则从文件hfmTree中读入），对文件ToBeTran中的正文进行编码，然后将结果存入文件CodeFile中。
void Solution::Encoding()
{
    fstream file;
    // check if tree exist
    if (root == NULL) {
        // read data from hfmTree.txt
        file.open("hfmTree.txt", ios::in);

        file >> n;
        arr.resize(n);
        weight.resize(n);

        for (int i = 0; i < n; ++i)file >> arr[i];
        for (int i = 0; i < n; ++i)file >> weight[i];

        file.close();

        root = BuildHuffmanTree();
    }

    file.open("ToBeTran.txt", ios::in);

    string text;
    file >> text;

    file.close();

    file.open("CodeFile.txt", ios::in | ios::out | ios::trunc);

    for (int i = 0; i < text.size(); ++i) {
        file << prefixCodes[text[i]];
    }

    file.close();

    cout << "------------------||   Encoding Process  ||------------------" << endl;

    cout << setw(40) << right << "Successfully encoded!" << endl;

    cout << "------------------|| huffmanCode Program ||------------------" << endl;

    system("pause");
```

```
}

// 利用已建好的哈夫曼树将文件CodeFile中的代码进行译码，结果存入文件TextFile中。
void Solution::Decoding()
{
    fstream file;
    // check if tree exist
    if (root == NULL) {
        // read data from hfmTree.txt
        file.open("hfmTree.txt", ios::in);

        file >> n;
        arr.resize(n);
        weight.resize(n);

        for (int i = 0; i < n; ++i)file >> arr[i];
        for (int i = 0; i < n; ++i)file >> weight[i];

        file.close();

        root = BuildHuffmanTree();
    }

    file.open("CodeFile.txt", ios::in);
    string code;
    file >> code;
    file.close();

    file.open("TextFile.txt", ios::in | ios::out | ios::trunc);

    TreeNode* cur = root;
    for (int i = 0; i < code.size(); ++i) {
        if (code[i] == '0')cur = cur->left;
        else cur = cur->right;

        // if leaf
        if (!cur->left && !cur->right) {
            file << cur->val;
            cur = root;
        }
    }

    file.close();
```

```
    cout << "------------------||   Decoding Process  ||------------------" << endl;

    cout << setw(40) << right << "Successfully decoded!" << endl;

    cout << "------------------|| huffmanCode Program ||------------------" << endl;

    system("pause");
}

// 将文件CodeFile以紧凑格式显示在终端上，每行50个代码。同时将此字符形式的编码文件写入文件CodePrin中。
void Solution::Print()
{
    // read from CodeFile
    fstream file;
    file.open("CodeFile.txt", ios::in);
    string code;
    file >> code;
    file.close();

    file.open("CodePrin.txt", ios::in | ios::out | ios::trunc);

    cout << "-------------||      CodeFile     ||-------------" << endl;
    for (int i = 0; i < code.size(); ++i) {
        if (i % 50 == 0) {
            cout << endl;
            if(i)file << endl;
        }
        cout << code[i];
        file << code[i];
    }
    cout << endl;
    cout << endl;
    cout << "-------------|| huffmanCode Program ||-------------" << endl;
    system("pause");
}

// 将已在内存中的哈夫曼树以直观的方式（树或凹入表形式）显示在终端上，同时将此字符形式的哈夫曼树写入文件TreePrint
中。
void Solution::TreePrinting()
{
    fstream file;
    // check if tree exist
    if (root == NULL) {
        // read data from hfmTree.txt
```

```
        file.open("hfmTree.txt", ios::in);

        file >> n;
        arr.resize(n);
        weight.resize(n);

        for (int i = 0; i < n; ++i)file >> arr[i];
        for (int i = 0; i < n; ++i)file >> weight[i];

        file.close();

        root = BuildHuffmanTree();
    }

    file.open("TreePrint.txt", ios::in | ios::out | ios::trunc);

    prettyPrintTree(root, tree, "", true);
    for (int i = 0; i < tree.size(); ++i) file << tree[i];

    file.close();

    system("pause");
}

TreeNode* Solution::BuildHuffmanTree()
{
    int n = arr.size();// get the size

    priority_queue<TreeNode*, vector<TreeNode*>, Compare> nodes;

    for (int i = 0; i < n; ++i)nodes.emplace(new TreeNode(arr[i], weight[i]));

    while (nodes.size() != 1) {
        auto left = nodes.top();
        nodes.pop();

        auto right = nodes.top();
        nodes.pop();

        auto newNode = new TreeNode('$', left->weight + right->weight);
        newNode->left = left;
        newNode->right = right;

        nodes.emplace(newNode);
```

```
    }

    // store codes
    storeCodes(nodes.top(), "");

    return nodes.top();
}

void Solution::calcAllNeeded()
{
    // read from ToBeTran
    fstream file;
    file.open("ToBeTran.txt", ios::in);
    string text;
    file >> text;

    // count each character
    for (int i = 0; i < text.size(); ++i)freq[text[i]]++;

    file.close();
}

void Solution::storeCodes(TreeNode* node, string prefix)
{
    if (!node)return;
    if (node->val != '$')
        prefixCodes[node->val] = prefix;
    storeCodes(node->left, prefix + '0');
    storeCodes(node->right, prefix + '1');
}

void Solution::prettyPrintTree(TreeNode* node, vector<string>& tree, string prefix, bool isLeft)
{
    if (node == nullptr) {
        cout << "Empty tree";
        return;
    }

    if (node->right) {
        prettyPrintTree(node->right, tree, prefix + (isLeft ? "│   " : "    "), false);
    }

    cout << prefix + (isLeft ? "└── " : "┌── ") + node->val + '[' + to_string(node->weight) + ']'
+ "\n";
```

	`tree.emplace_back(prefix + (isLeft ? "└── " : "┌── ") + node->val + '[' + to_string(node->weight) + ']' + "\n");`
	
	`if (node->left) {`
	`prettyPrintTree(node->left, tree, prefix + (isLeft ? "    " : "│   "), true);`
	`}`
	`}`

实验总结	本次实验不仅加深了对哈夫曼编码的理解，更是巩固了对树这一数据结构的基础认识，体会到了树结构在计算科学中的实际应用，哈夫曼编码作为一种无损数据压缩技术，其利用的树的原理十分巧妙，先根据数据建立树，然后将树作为其编码工具对数据进行编码，同时以相同的树作为解码工具，通过遍历操作进行解码，学习这一实用工具使人受益匪浅。