

PA3实验报告

1. 涉及数据结构和相关背景

- 二叉树
 - 二叉树的数组模拟（二叉树存储方式）
 - 二叉树寻找子节点与父节点
 - 二叉树的建立
- 哈夫曼树
 - 哈夫曼树的建立
 - 哈夫曼树的编码方式
 - 解码方式
- 文件输入输出
 - 文件指针
 - 输入输出流
 - 二进制文件与文本文件
- 文本格式
 - ascii码
 - 扩展ascii码

2. 实验内容

2.1 问题描述

- 实现对ASCII字符文本进行Huffman压缩，并且能够进行解压。
将给定的文本文件使用哈夫曼树进行压缩，并解压。

2.2 基本要求

- 基本要求为：能将文本文件压缩、打印压缩后编码、解压压缩后的文件。对实际使用的具体数据结构除必须使用二叉树外不做要求。
- 用一个二叉树表示哈夫曼树，因为ASCII表一共只有127个字符，可以直接使用数组来构造Huffman树。
leftChild和rightChild分别表示当前结点左儿子和右儿子的下标。因为1到127分别表示与ASCII表的符号相对应，所以这里无需记录每个结点代表什么符号。

```
struct Node
{
    int leftChild;

    int rightChild;

} tree[256];
```

- 而在解压的时候，由于不知道每个结点实际表示什么符号，所以需要在树上记录下。

```
struct Nodev
{
    int leftChild;

    int rightChild;

    char c;

} tree[256];
```

2.3 数据结构设计

- 用二叉树来实现哈夫曼树结构
- 存储树结构利用数组对树的结构进行模拟

```
//Haffman tree Node
struct Node
{
    int weight;           //权重
    char data;            //字符信息
    int leftChild;        //左孩子
    int rightChild;       //右孩子
    int parent;           //父节点
}tree[512];
```

为更加快速准确找到父节点，这里用parent记录父节点的指针，weight记录权重或者权重和（这里相应的是字母出现的频率）

- 哈夫曼树
 - 优点
 - 保证概率大的符号对应于短码，概率小的符号对应于长码而且所有的短码得到充分利用；
 - 且每次缩减信源的最后两个码字总是最后一位不同，前面各位相同，这两个特点保证了哈夫曼编码一定是最佳的。
 - 虽然哈夫曼编码构造出来的码不唯一，但是其平均码长是相等的，所以不影响编码效率和数据压缩性能。
 - 缺点

- 如果对单个字母进行编码，平均码字长可能还与理论上的最优编码率还有一定差距，哈夫曼编码算法是从上而下构造树。当信源符号集很大时，这种方法不方便；、
- 从信道传输上来看，对应的长码一旦产生误码，某个码字的前缀部分可能成为另一个码字而发生误差，并导致错误后传。

• 压缩方式

- 进行字符频率的统计，并进行排序（由小到大），方便后序建树

```

■   const int N = 256;

    //frequency table
    int cnt[N] = { 0 };

```

- 进行哈夫曼树的构建并保存
- 从叶子节点开始遍历到根节点得到每一个字符所对应的哈夫曼编码，记录到编码表中

```

■   //mapping table
    map<unsigned char, string> mp;

```

- 比照编码表遍历原文件，将哈夫曼编码输出
- 每8位将哈夫曼编码转换为char类型字符，输出到压缩文件中（不足8位的补充1）

• 解压缩方式

- 通过读入压缩文件，将压缩文件中的字符每8位转换成相应的二进制编码，（并剪掉末尾补充的1）得到原来的哈夫曼编码
- 通过遍历原来哈夫曼树到叶子节点来确定每一段哈夫曼编码对应的字符值
 - 这里是时间复杂度较小的方式，也可以通过对表的查询遍历来进行哈夫曼编码的解码，但时间复杂度相对较高
- 将字符值输出到相应文件中得到解压后的文件

• 其他格式文件的压缩与解压缩

- 纯英文字符
 - 仅用到 `ascii` 码中0 ~ 127，`char`类型对应的整形值为正数
- 中文字符
 - 涉及到多位记录一个字符，`char`类型对应的整型值为负数
 - 此处用`unsigned char`记录可以将其转换为256以内的正数
- 二进制文件
 - 所有文件都可以进行二进制读入与二进制输出，包括图片(`jpeg`, `bmp` ...), 编码格式文档(`word`, `pdf`, `pptx`, `doc`, `docx` ...), 以及纯文本文件等
 - 则可以进行二进制读入与输出保证可压缩的文件种类可以达到最大化

2.4 功能说明

- 读入文件

```
void readFile(string path)
{
    /*
    param : path 读取文件路径
    function: 读取文件
    */
    FILE* fp; //文件指针
    fp = fopen(path.c_str(), "rb"); //read the file in biary form

    unsigned char ch; //保证压缩文件的多样性此处用uc来接字符
    while (!feof(fp)) //文件指针获取字符
    {
        ch = fgetc(fp);
        cnt[ch + 0]++; //统计每个字符出现的频次
    }

    fclose(fp);
}
```

- 注意此处是二进制的读入文件，保证除了文本文件可以进行压缩之外其余形式的文件同样可以被压缩
- 此处接字符为了中文以及二进制文件，用unsigned char进行读入
- 读入的过程中同时进行字符频次的统计

- 初始化哈夫曼树（进行叶子节点的初始化）

```
//_Pred of tree[]
bool cmp(const Node& a, const Node& b)
{
    /*
    tree叶子节点的排序谓词
    */
    return a.weight < b.weight;
}

//count for all leaf nodes
int ptr = 0; //tree中叶子节点的个数

//Haffman tree leaves
void placeTreeLeaves()
{
    /*
    function : 在树中初始化节点并进行排序
    */
    for (int i = 0; i < N; i++)
    {
```

```

        if (cnt[i]) //读入有频次的节点
        {
            Node* nd = new Node;
            nd->data = (unsigned char)i; //char(i);
            nd->leftChild = nd->rightChild = nd->parent = -1;
            nd->weight = cnt[i];
            tree[ptr++] = *nd;
        }
    }
    sort(tree, tree + ptr, cmp); //初始就对叶子节点按照频
    次排序
}

```

进行叶子节点的排序和初始化

排序是为了在最初进行构建时寻找最小的两个叶子节点时候可以减少一部分时间

- 建立哈夫曼树

```

//select the two smallest position
pair<int, int> select(int l, int r)
{
    /*
    param : l, r tree的区间左右端点
    function : 选择所在区间内的最小权重所在位置次小元素所在位置
    */
    int min1 = -1;
    int min2 = -1;

    int min1_w = 0x7fffffff;
    int min2_w = 0x7fffffff;

    for (int i = l; i < r; i++)
    {
        if (tree[i].parent == -1)
        {
            if (tree[i].weight < min1_w)
            {
                //update the smallest weight
                min2_w = min1_w;
                min1_w = tree[i].weight;

                //update index
                min1 = min2;
                min2 = i;
            }

            else if (tree[i].weight < min2_w)
            {
                min2_w = tree[i].weight;
                min1 = i;
            }
        }
    }
}

```

```

    }
    }
}

return { min1, min2 };
}

//Haffman encoding
void setUpHaffmanTree()
{
    /*
    function : 建立哈夫曼树，更新哈夫曼树中节点的孩子和父母关系
    */
    int m = 2 * ptr - 1; //建立n个节点的二叉树（哈夫曼树），没有单叶子节点，需要2 * n
    - 1个节点就可以
    for (int i = ptr; i < m; i++)
    {
        pair<int, int> childs = select(0, i); //pair记录最小的两个节点位置

        //info for non-leaf nodes
        tree[i].leftChild = childs.first; //更新新的根节点的子节点的信息
        tree[i].rightChild = childs.second;
        tree[i].weight = tree[childs.first].weight +
        tree[childs.second].weight;
        tree[i].parent = -1;

        //info for leaf nodes
        tree[childs.first].parent = i; //更新父节点的信息
        tree[childs.second].parent = i;
    }
}

```

哈夫曼树的建立过程就是

- 在频率作为权重的树节点中选择权重最小的两个节点作为新节点的两个子节点
- 将这三个节点的关系设置好
- 将新节点的权值赋值为两个子节点的权值之和
- 将新的子节点加入到树中
- 原来的两个节点不参与新一轮的选择
- 重新在树中选择两个最小权重的子节点，进行下一轮操作
- 循环m - ptr次（所有的非叶子节点循环完）这时就建树完成了

例如以下的 test1.txt 文件

```
test1.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

When,in disgrace with fortune and men's eyes
I all alone beweeep my outcast state
And trouble deaf heaven with my bootless cries
And look upon myself and curse my fate
Wishing me like to one more rich in hope
Featured like him, like him with friends possess'd
Desiring this man's art and that man's scope,
With what I most enjoy contented least
Yet in these thoughts myself almost despising
Haply I think on the,and then my state
Like to the lark at break of day arising
From sullen earth, sings hymns at heaven's gate
For thy sweet love remember'd such wealth brings
That then I scorn to change my state with kings.
```

第 14 行, 第 27 列 100% Windows (CRLF) UTF-8

哈夫曼树建立为

```
Please enter the file to
Compress:
test1.txt
0 1 39 -1 -1
1 1 39 -1 -1
2 1 40 -1 -1
3 1 40 -1 -1
4 1 41 -1 -1
5 1 41 -1 -1
6 1 42 -1 -1
7 1 42 -1 -1
8 2 43 -1 -1
9 3 45 -1 -1
10 3 46 -1 -1
11 3 46 -1 -1
12 4 47 -1 -1
13 5 48 -1 -1
14 6 49 -1 -1
15 6 50 -1 -1
16 7 51 -1 -1
17 7 51 -1 -1
18 8 52 -1 -1
19 9 53 -1 -1
20 9 53 -1 -1
21 10 54 -1 -1
22 11 55 -1 -1
23 13 56 -1 -1
24 13 57 -1 -1
25 13 57 -1 -1
26 15 58 -1 -1
27 19 60 -1 -1
28 20 61 -1 -1
29 21 61 -1 -1
30 23 63 -1 -1
31 31 64 -1 -1
32 31 65 -1 -1
33 35 66 -1 -1
34 38 66 -1 -1
35 42 68 -1 -1
36 49 69 -1 -1
37 63 70 -1 -1
38 98 73 -1 -1
39 2 43 1 0
40 2 44 3 2
41 2 44 5 4
42 2 45 7 0
43 4 47 39 8
44 4 48 41 40
45 5 49 9 42
46 6 50 11 10
47 8 52 43 12
48 9 54 13 44
49 11 55 14 45
50 12 56 46 15
51 14 58 17 16
52 16 59 47 18
53 18 59 20 19
54 19 60 21 48
55 22 62 49 22
56 25 62 23 50
57 26 63 25 24
58 29 64 26 51
59 34 65 53 52
60 38 67 54 27
61 41 67 28 28
62 47 68 56 55
63 54 69 30 57
64 60 70 31 58
65 65 71 59 32
66 73 71 34 33
67 79 72 61 60
68 89 72 62 35
69 102 73 63 36
70 123 74 37 64
71 138 74 66 65
72 168 75 68 67
73 200 75 69 38
74 261 76 71 70
75 368 76 73 72
76 629 -1 75 74
```

其中-1代表没有子节点或者没有父节点

- 建立哈夫曼编码查询表

```
//create reference table (map)
void encoding()
{
    /*
    function : 建立哈夫曼编码对应表
    */
    for (int i = ptr - 1; i >= 0; i--)
    {
        string s = "";
        unsigned char ch = tree[i].data;
        int child = i;
        while (tree[child].parent != -1) //倒退到根节点
        {
            int parent = tree[child].parent;
            if (tree[parent].leftChild == child) //左孩子编码0
            {
                s = '0' + s;
            }
            else if (tree[parent].rightChild == child) //右孩子编码1
            {
                s = '1' + s;
            }
            child = parent;
        }
        mp.insert({ ch, s }); //建立编码查询表
        mp_decoding.insert({ s, ch });
        encoded[cntRef++] = s; //encoded[]记录所有哈
    }
}
```

夫曼编码

哈夫曼编码过程

- 从叶子节点遍历到根节点
- 当前节点是父节点的左孩子：编码前加上0
- 当前节点是父节点的右孩子：编码前加上1
- 遍历到根节点的时候得到对应字符的哈夫曼编码
- 遍历每一个叶子节点

同样的 test1.txt 文件，哈夫曼编码为


```

-1 101010001
106 101010000
46 011101111
68 011101110
72 011101101
76 011101100
84 010010111
89 010010110
65 10101001
70 01001010
87 01000101
118 01000100
73 1010101
44 0111010
39 0100100
98 0100011
102 1111111
112 1111110
119 101011
107 101001
117 101000
99 011100
103 010011
10 010000
13 000011
121 000010
100 11110
108 01111
109 01101
114 01100
111 00000
104 1110
105 1011
97 1001
110 1000
115 0101
116 0001
101 110
32 001

```

其中左侧为 `ascii` 码值(其中有负数可能是有某个中文符号)，右侧为哈夫曼编码，从频率最低到最高

可以发现频率越高，哈夫曼编码越短，便于查询

- 创建哈夫曼编码文件

```

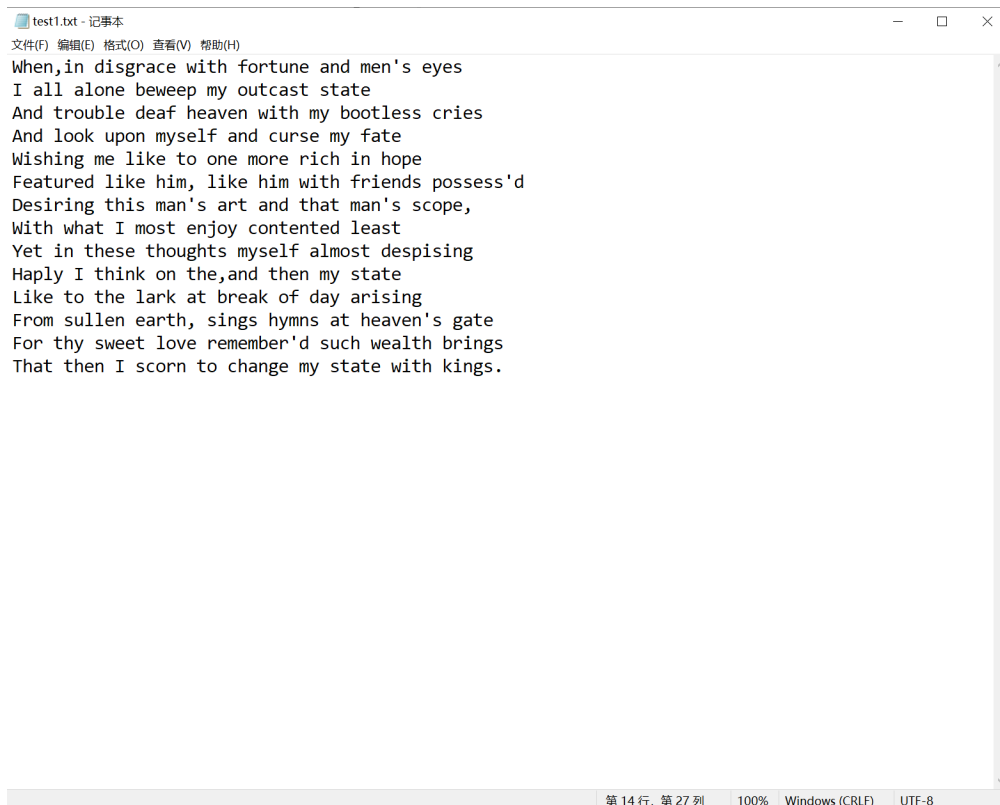
//create encoded.txt file
void makeCompressedFile(string readPath, string tarPath)
{
    /*
    function : 文件展示哈夫曼编码
    */
    FILE *fout, *fin;
    fout = fopen(readPath.c_str(), "rb");           //二进制写入
    fin = fopen(tarPath.c_str(), "w");              //文本文件输出（便于观察）
    while (!feof(fout))
    {
        fputs(mp[fgetc(fout)].c_str(), fin);        //查询编码表，相应编码输出到文件中记录
    }

    fclose(fout);
    fclose(fin);
    cout << endl << "Successfully compressed." << endl;
}

```

为方便展示过程，此处将哈夫曼编码输出到一个名为 `encoded.txt` 的文件当中，便于观察相应的哈夫曼编码

- 例如以下的 `test1.txt` 文件



```
test1.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
When,in disgrace with fortune and men's eyes
I all alone beweep my outcast state
And trouble deaf heaven with my bootless cries
And look upon myself and curse my fate
Wishing me like to one more rich in hope
Featured like him, like him with friends possess'd
Desiring this man's art and that man's scope,
With what I most enjoy contented least
Yet in these thoughts myself almost despising
Haply I think on the,and then my state
Like to the lark at break of day arising
From sullen earth, sings hymns at heaven's gate
For thy sweet love remember'd such wealth brings
That then I scorn to change my state with kings.
第 14 行, 第 27 列 100% Windows (CRLF) UTF-8
```

- 经过哈夫曼编码产生的 `encoded.txt` 文件为



```
encoded.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
01000101111011010000111010101110000011111010110101010011011001001011100110001101011101100011110
00111111100000011000001101000100011000110011000111100010110111010000100100010100111000001011001
0100001101000010101010011100111101110011001011110000010001100010100011110101011101101111000
101101000010001000001010000001011100100101010001001010001100100011100000110100001010100110001
111000100010110000001010000100011011111000111101101001111110011101101001010001001101000001
101011101100011110001011010000100010100011000000000001011111001010100101110001100101111001
0100001101000010101001100011110001011110000000001010010011010001111100000100100110100001001
011100111111111001100110001111000101110010100001100010111000101101000010001111111001000111000
00110100000100010110110101110101110000100110010111000101111011010011100010001000000010000
01000110001011010000001100110001011001011100111000101110000011110010000011010000011010000
01001010110100100011010000110011011100010111011010011100011101011010111010001011110111
0100111000111101011010011010110110001111000111110110010111010001111100000001
01010111001010101010010011110000011010000011101110110010110110110001011001000111101011
010100101101100110000100100010100110010110000010011001100011110001000111101001000100110110011
0000100100010100101011100000011110110011010000011010000010001011010001111000110101111010
01000100110101010110100000010100010011101000101010000000000010001011100000010000001110100
000011101110001011111010010101000100001101000001001011011000010011011100000100011110110010111
000100011110000001010000100111110000101010010110100001001011100111111111001100101110110100000
0101000100111110110010111110101101110000100110000110100000111011011001111100111100001000
1101010100100011110101110001010010010000010000100011110110011101010011000111100010001111011010
000101101000010001010100011001000111000001101000001101100101110100111000100010000000100011110
1100010111110010110010100100110010001001010001101100110100100100000111110011111010010000
10001100101100101101101110000100110000110100000100101001100000000110100101011010000111101111
10100000111010010110000011110011101000101011100001001101010011110000010011011000010100110010
001001111011010010100010011010000100100010100111001000111000001101000001001010000000110000
10001111000001000101011010111101100001001011100000100010011000101100110011011001101010001111
001100010010011110001010110100001110011100011010111010010111000111100010100011011001011100001
0011010100001101000001001011111101001000100100011110110100000110101010010101111000000001100100
00010001000000010111001110100110000100111100010110100001000101010001100100011100011010111011000
11110001101001101110000100110101011101010001
第 1 行, 第 1741 列 100% Windows (CRLF) UTF-8
```

- 进行文件压缩

```

int redundant = 0; //记录尾部多余的0
string fc = "";
//change the encoding to character
void encodingPress(string cmPath)
{
    /*
    param : cmPath 哈夫曼编码展示文件
    function : 每8位将压缩后的文件转化为相应字符，达到压缩的效果
    */

    string temp_path = "pro.txt"; //将压缩后的文件命名为pro.txt
    int temp = 0;

    FILE* fout, * fin;
    fout = fopen(cmPath.c_str(), "rb"); //同样是二进制读入
    fin = fopen(temp_path.c_str(), "w");

    unsigned char ch;
    int cnt = 0; //8位计数器
    while (!feof(fout))
    {
        temp = (fgetc(fout) - '0') ? temp + fps(2, 7 - cnt) : temp; //计算8位对应的ascii码的值是多少
        cnt++;

        if (cnt == 8) //每8个进行一次输出
        {
            fputc((unsigned char)temp, fin);
            fc += (unsigned char)temp;
            temp = 0;
            cnt = 0;
        }
    }

    if (cnt) //如果最后需要补位
    {
        while (cnt != 8)
        {
            redundant++;
            cnt++;
            temp += fps(2, 8 - cnt);
        }

        fputc((unsigned char)temp, fin);
        fc += (unsigned char)temp;
    }

    fclose(fout);
    fclose(fin);
}

```

- 通过读入哈夫曼编码每8位计算 `ascii` 码，输出压缩后的字符
- 此处计算 `ascii` 码的方法是

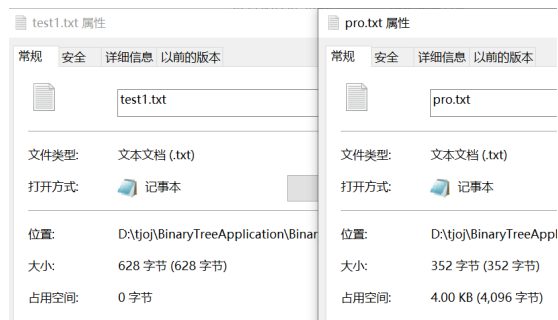
- 左边为高位
- 每8位从高到低计算2的幂次累加和
- 计算幂次利用快速幂

```
int fps(int a, int p)
{
    /*
    function : 计算幂次
    */
    int ans = 1;
    while (p)
    {
        if (p & 1) ans *= a;
        a *= a;
        p >>= 1;
    }
    return ans;
}
```

- 特别注意补充位数的问题，因为最后可能不足8位，需要进行位数补充
 - 最后记录下补充的位数(在解码时需要进行相应修剪)，此处用 `int redundant` 进行记录
- 同样以 `text1.txt` 为例，最后输出的 `pro.txt` 压缩文件内容为



- 可以看到均为乱码，但是空间相较于原文件缩小近一半(56%)



- 解码还原 `encoded.txt` (哈夫曼编码文件)
 - 将压缩文件字符转换为其二进制字符串形式

```
//convert a char to its binary form string
string charTobChar(unsigned char ch)
{
    /*
    function : 将字符转换为其二进制字符串形式
    */
    string s = "00000000";

    for (int i = 7; i >= 0; i--) //将字符对应二进制不断右移观察最后一位是否有值，对初始字符串相应位置进行更新
    {
        if (ch & 1) s[i] = '1';
        ch >>= 1;
    }
    return s;
}
```

- 重新构建 `encoded.txt` 文件，输出到 `encoding.txt` 文件中

```
//reshape encoding file
void makeEncodingFile(string pressed)
{
    /*
    function : 将压缩后的文件转化为哈夫曼编码文件
    */
    ofstream tar; //文件输出流打开
    tar.open("encoding.txt", ios::out);

    string temp = "";

    for (int i = 0; i < fc.size(); i++)
    {
        temp += charTobChar(fc[i]);
    }
    cout << endl;

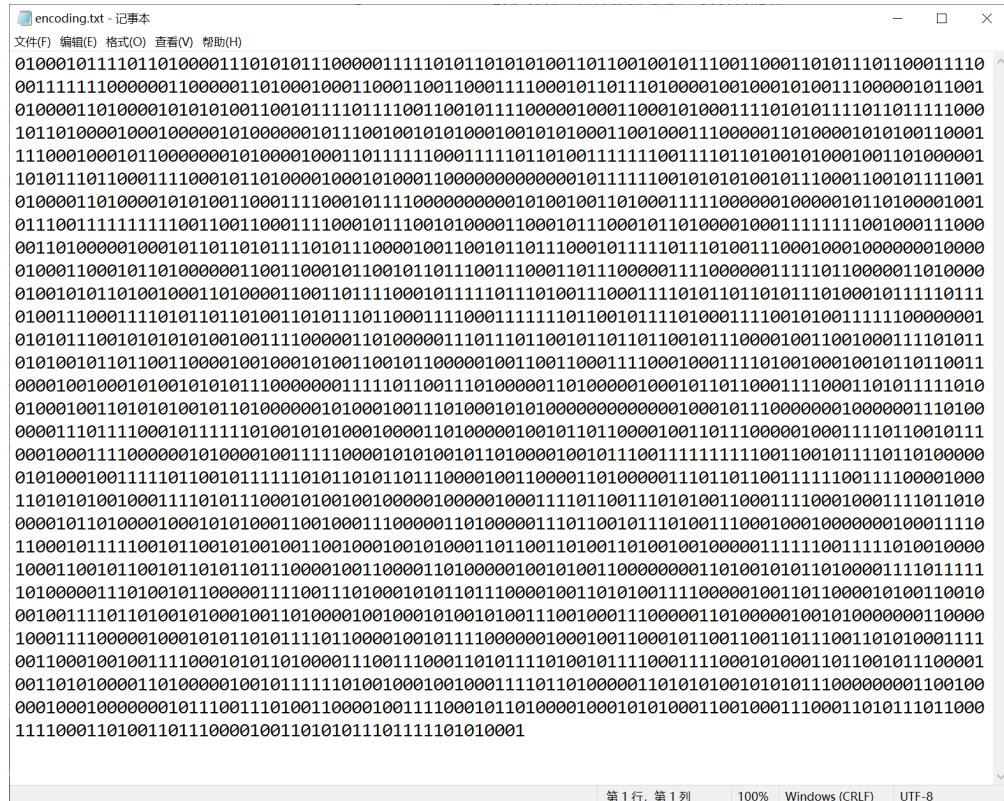
    temp = temp.substr(0, temp.length() - redundant - 1);
}
```

```
tar << temp;
tar.close();
}
```

//临时字符串输入到新文件中

此处的从压缩文件读取字符的方法试验很多次以及多种方法没有成功，最后采用全局变量记录压缩后的字符串的方式来进行从压缩文件读取字符的模拟

同样以 test1.txt 为例，转换后的 encoding 文件与之前的 encoded 文件完全相同



- 进行解码成 decoded.txt 文件

```
//Huffman decoding
void decoding(string readPath, string tarPath)
{
    /*
    param: readPath 从encoding.txt文件读入的哈夫曼编码
    param: tarPath 目标解压文件路径

    */
    FILE* fout, * fin;
    fout = fopen(readPath.c_str(), "rb");
    fin = fopen(tarPath.c_str(), "wb");

    string temp = "";
    while (!feof(fout))
    {
        temp += fgetc(fout);
        if (check(temp))
        {
            fputc(mp_decoding[temp], fin);
        }
    }
}
```

//二进制读入

//二进制输出

//这里利用查表的方式进行解码

```

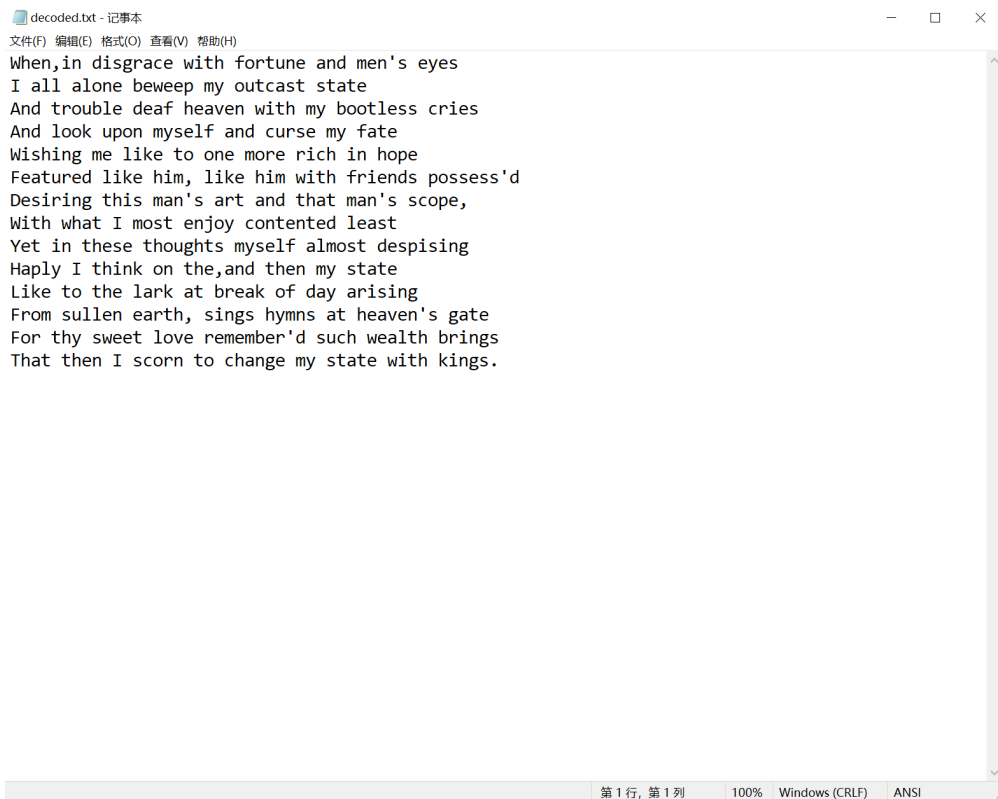
        temp = "";
    }

}

fclose(fout);
fclose(fin);
}

```

- 注意二进制输入，二进制输出
- 进行解码的时候可以构建哈夫曼树进行解码，同样也可以保存最开始构建的哈夫曼树进行解码
- 同样 test1.txt 文件的解码



decoded.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

When,in disgrace with fortune and men's eyes
 I all alone beweeep my outcast state
 And trouble deaf heaven with my bootless cries
 And look upon myself and curse my fate
 Wishing me like to one more rich in hope
 Featured like him, like him with friends possess'd
 Desiring this man's art and that man's scope,
 With what I most enjoy contented least
 Yet in these thoughts myself almost despising
 Haply I think on the,and then my state
 Like to the lark at break of day arising
 From sullen earth, sings hymns at heaven's gate
 For thy sweet love remember'd such wealth brings
 That then I scorn to change my state with kings.

第 1 行, 第 1 列 100% Windows (CRLF) ANSI

- 获取文件扩展名

```

//draw file extension from the given path
string drawExtension(string path)
{
    /*
    param: path 文件路径
    function : 获取文件拓展名
    */
    string temp = "";
    for (int i = path.size() - 1; i >= 0; i--)
    {

```

```

        if (path[i] == '.')
        {
            temp = '.' + temp;
            break;
        }
        else temp = path[i] + temp;

    }

    return temp;
}

```

自动获取文件扩展名，用于构建解压文件

- 主函数调用

```

int main()
{
    string path;
    cout << "Please enter the file to Compress:" << endl;
    cin >> path;

    readFile(path);                                //读取文件

    placeTreeLeaves();                             //初始化哈夫曼树

    setUpHaffmanTree();                            //建立哈夫曼树

    encoding();                                    //哈夫曼编码

    string cmPath = "encoded.txt";

    makeCompressedFile(path, cmPath);               //展示哈夫曼编码文件

    encodingPress(cmPath);                         //创建压缩文件

    makeEncodingFile("pro.txt");                   //从压缩文件还原哈夫曼编码展示文件

    int choice = 0;
    cout << "decode the compressed file?" << endl << "1 : YES" << endl << "0
: NO" << endl;
    cin >> choice;
    if (choice)
    {
        string decPath = "decoded" + drawExtension(path); //创建解压
文件
        decoding("encoding.txt", decPath); //还原最初源文件
    }

    return 0;
}

```


2.5 调试分析

- 引导界面

```
Microsoft Visual Studio 调试控制台
Please enter the file to Compress:
1.pptx

Successully compressed.

decode the compressed file?
1 : YES
0 : NO
1
```

- 展示哈夫曼树

```
Microsoft Visual Studio 调试控制台
16 7 51 -1 -1
17 7 51 -1 -1
18 8 52 -1 -1
19 9 53 -1 -1
20 9 53 -1 -1
21 10 54 -1 -1
22 11 55 -1 -1
23 13 56 -1 -1
24 13 57 -1 -1
25 13 57 -1 -1
26 15 58 -1 -1
27 19 60 -1 -1
28 20 61 -1 -1
29 21 61 -1 -1
30 28 63 -1 -1
31 31 64 -1 -1
32 31 65 -1 -1
33 35 66 -1 -1
34 38 66 -1 -1
35 42 68 -1 -1
36 48 69 -1 -1
37 63 70 -1 -1
38 98 73 -1 -1
39 2 43 1 0
40 2 44 3 2
41 2 44 5 4
42 2 45 7 6
43 4 47 39 8
44 4 48 41 40
45 5 49 9 42
46 6 50 11 10
47 8 52 43 12
48 9 54 13 44
49 11 55 14 45
50 12 56 46 15
51 14 58 17 16
52 16 59 47 18
53 18 59 20 19
54 19 60 21 48
55 22 62 49 22
56 25 62 23 50
57 26 63 25 24
58 29 64 26 51
59 34 65 53 52
60 38 67 54 27
61 41 67 29 28
62 47 68 56 55
63 54 69 30 57
64 60 70 31 58
65 65 71 59 32
66 73 71 34 33
67 79 72 61 60
68 89 72 62 35
69 102 73 63 36
70 123 74 37 64
71 138 74 66 65
72 168 75 68 67
73 200 75 69 38
74 261 76 71 70
75 368 76 73 72
76 629 -1 75 74
```

- 展示哈夫曼编码

```

Successfully compressed.
-1 101010001
106 101010000
46 011101111
68 011101110
72 011101101
76 011101100
84 010010111
89 010010110
65 10101001
70 01001010
87 01000101
118 01000100
73 1010101
44 0111010
39 0100100
98 0100011
102 111111
112 111110
119 101011
107 101001
117 101000
99 011100
103 010011
10 010000
13 000011
121 000010
100 11110
108 01111
109 01101
114 01100
111 00000
104 1110
105 1011
97 1001
110 1000
115 0101
116 0001
101 110
32 001

decode the compressed file?
1 : YES
0 : NO
1

```

- 文件类型与文件压缩率
 - 对纯英文文本文件压缩

test1.txt





test1.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

When,in disgrace with fortune and men's eyes
 I all alone beweepe my outcast state
 And trouble deaf heaven with my bootless cries
 And look upon myself and curse my fate
 Wishing me like to one more rich in hope
 Featured like him, like him with friends possess'd
 Desiring this man's art and that man's scope,
 With what I most enjoy contented least
 Yet in these thoughts myself almost despising
 Haply I think on the,and then my state
 Like to the lark at break of day arising
 From sullen earth, sings hymns at heaven's gate
 For thy sweet love remember'd such wealth brings
 That then I scorn to change my state with kings.

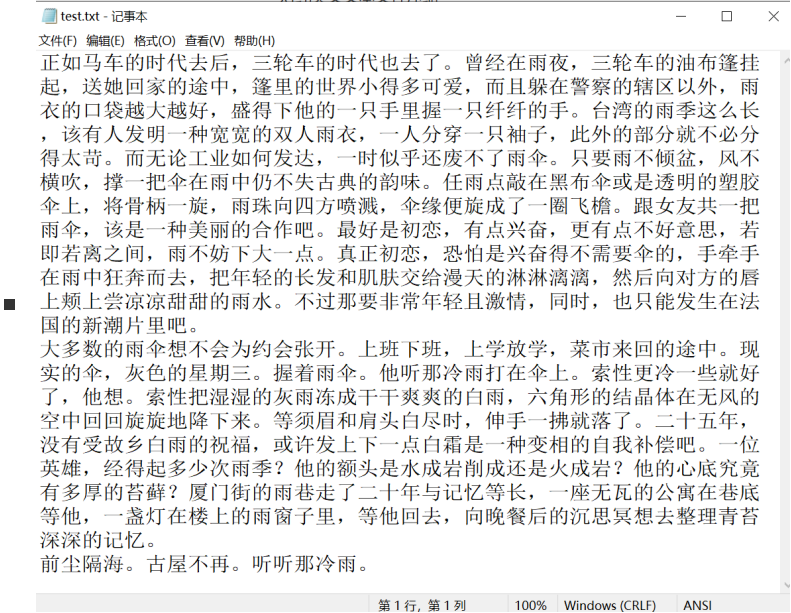
第 1 行, 第 1 列 100% Windows (CRLF) UTF-8

压缩率 56%



test1.txt 属性		pro.txt 属性	
常规		常规	
 test1.txt		 pro.txt	
文件类型: 文本文档 (.txt)		文件类型: 文本文档 (.txt)	
打开方式:  记事本		打开方式:  记事本	
位置: D:\tj\oj\BinaryTreeApp		位置: D:\tj\oj\BinaryTreeApp	
大小: 628 字节 (628 字节)		大小: 352 字节 (352 字节)	
占用空间: 0 字节		占用空间: 4.00 KB (4,096 字节)	

对中文文本文件压缩

test.txt



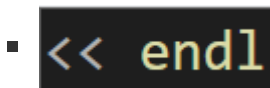
压缩率 75.7%

test.txt 属性		pro.txt 属性	
常规		常规	
 test.txt		 pro.txt	
文件类型: 文本文档 (.txt)		文件类型: 文本文档 (.txt)	
打开方式:  记事本		打开方式:  记事本	
位置: D:\tj\oj\BinaryTreeApp		位置: D:\tj\oj\BinaryTreeApp	
大小: 1.32 KB (1,356 字节)		大小: 1.00 KB (1,026 字节)	
占用空间: 4.00 KB (4,096 字节)		占用空间: 4.00 KB (4,096 字节)	

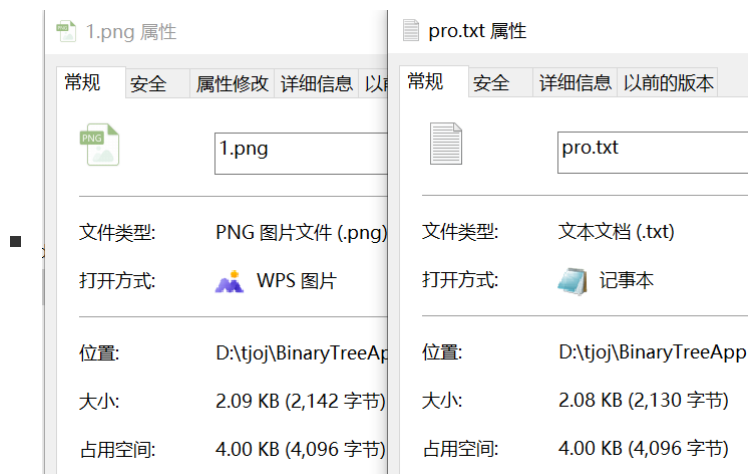
对图片文件压缩

对 .png 文件压缩

1.png

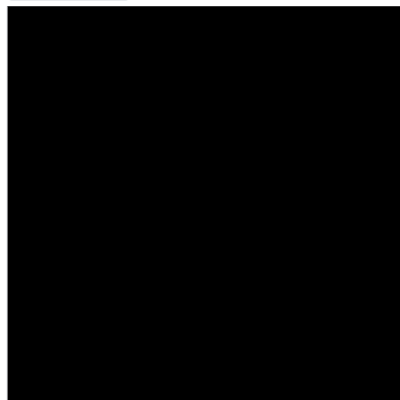


压缩率 99.4%

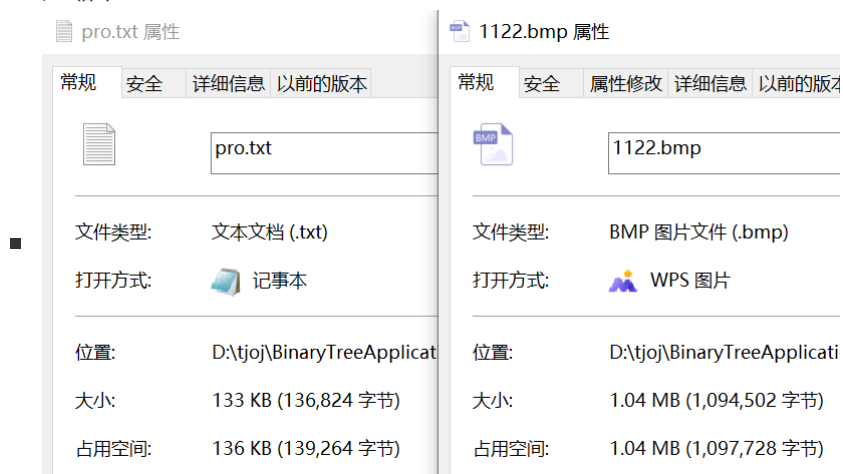


■ 对 .bmp 文件压缩 (纯色)

■ 1122.bmp



■ 压缩率 12.5%

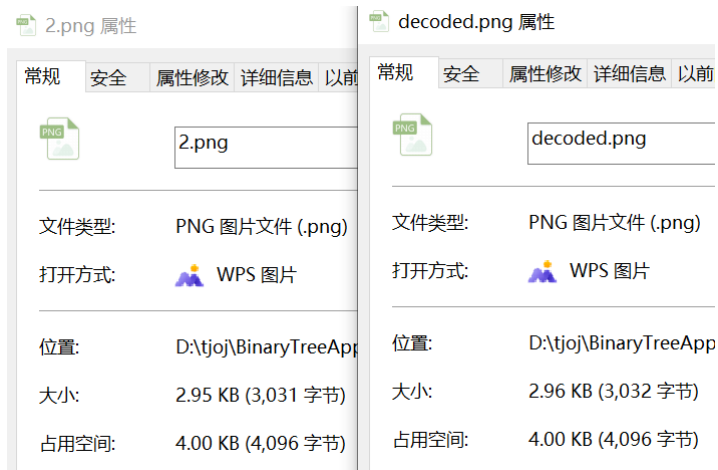


■ 对 .bmp 文件压缩 (彩色)

■ 2.png



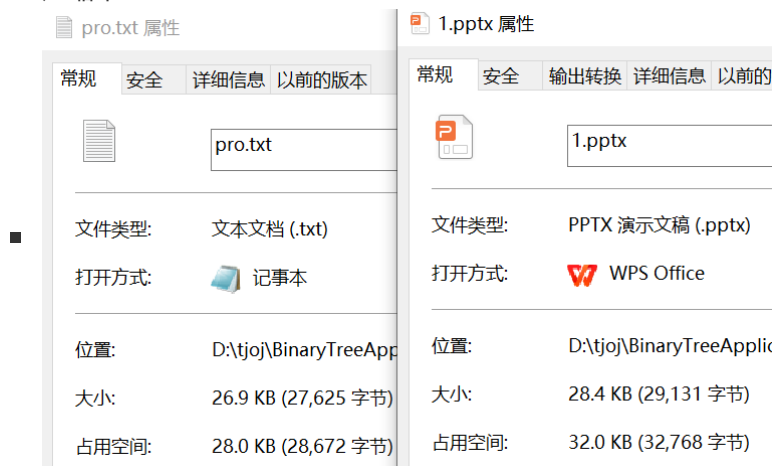
■ 压缩率 99.96%



- 图片影响
 - 相对来说纯色与 bmp 的压缩率更高

- 对pptx文件压缩

- 1.pptx
- 压缩率 94.83%



- 压缩文件字符读取
 - 解压时通过多次调试，尝试了多种读取方式
 - 方式1 `stringstream` 流读取到字符串中

```
#include <sstream>
...
string fc;
ifstream inputStream;
stringstream strStream;
inputStream.open(pressed);
strStream << inputStream.rdbuf();
inputStream.close();
fc = strStream.str();
```

- 方式2 文件指针

```
FILE* fp;
fp = fopen(pressed.c_str(), "r");

int cnt = 0;
while (!feof(fp))
{
    cnt++;
    fc += fgetc(fp);
}
```

- 方式3 流迭代器？

```
fstream fin;
fin.open(pressed);
istreambuf_iterator<char> beg(fin), end;

string fc(beg, end);
fin.close();
```

- 但都只能读取部分文件，最后选择了进行全局变量字符串存储

- `unsigned char`
 - `unsigned char` 可以读取扩展 `ascii` 码(中文字符)
 - 进行频率记录时要把数组开到256，防止转换乱码

3. 实验总结

- 二叉树/哈夫曼树
 - 熟悉二叉树的建立
 - 熟悉哈夫曼树的建立
 - 熟悉哈夫曼编码的产生过程
 - 哈夫曼压缩过程
 - 进行哈夫曼树的重建与解码
 - 哈夫曼编码的局限性
- 文件读写
 - 文件指针
 - 输入输出流
 - 熟悉文本格式、二进制输入输出

4. 源代码

HaffmanTree.cpp

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <stdio.h>
#include <map>
#include <algorithm>
#include <iomanip>

using namespace std;
const int N = 256;

//frequency table
int cnt[N] = { 0 };

//Haffman tree Node
struct Node
{
    int weight;
    char data;
    int leftChild;
    int rightChild;
    int parent;
}tree[512];

//count for all leaf nodes
int ptr = 0;

//mapping table
map<unsigned char, string> mp;
map<string, unsigned char> mp_decoding;

//decoding reference
int cntRef = 0;
string encoded[N];

//content of compressed file
string fc = "";

//read file
void readFile(string path)
{
    FILE* fp;
    fp = fopen(path.c_str(), "rb"); //read the file in binary form

    unsigned char ch;
    while (!feof(fp))
    {
        ch = fgetc(fp);
```

```

        //cout << ch + 0<< " ";
        cnt[ch + 0]++;
    }
    cout << endl << endl;;

    fclose(fp);
}

//_Pred of tree[]
bool cmp(const Node& a, const Node& b)
{
    return a.weight < b.weight;
}

//Haffman tree leaves
void placeTreeLeaves()
{
    for (int i = 0; i < N; i++)
    {
        if (cnt[i])
        {
            Node* nd = new Node;
            nd->data = (unsigned char)i;
//char(i);
            nd->leftchild = nd->rightchild = nd->parent = -1;
            nd->weight = cnt[i];
            tree[ptr++] = *nd;
        }
    }
    sort(tree, tree + ptr, cmp);
}

//select the two smallest position
pair<int, int> select(int l, int r)
{
    int min1 = -1;
    int min2 = -1;

    int min1_w = 0x7fffffff;
    int min2_w = 0x7fffffff;

    for (int i = l; i < r; i++)
    {
        if (tree[i].parent == -1)
        {
            if (tree[i].weight < min1_w)
            {
                //update the smallest weight
                min2_w = min1_w;
                min1_w = tree[i].weight;

                //update index
                min1 = min2;
            }
        }
    }
}

```



```

        min2 = i;

    }

    else if (tree[i].weight < min2_w)
    {
        min2_w = tree[i].weight;
        min1 = i;
    }
}

return { min1, min2 };
//cout << min1 << " " << min2;
}

//Haffman encoding
void setUpHaffmanTree()
{
    int m = 2 * ptr - 1;
    for (int i = ptr; i < m; i++)
    {
        pair<int, int> childs = select(0, i);

        //info for non-leaf nodes
        tree[i].leftChild = childs.first;
        tree[i].rightChild = childs.second;
        tree[i].weight = tree[childs.first].weight + tree[childs.second].weight;
        tree[i].parent = -1;

        //info for leaf nodes
        tree[childs.first].parent = i;
        tree[childs.second].parent = i;

    }
}

//find a sequence of results of 2 squares
int fps(int a, int p)
{
    int ans = 1;
    while (p)
    {
        if (p & 1) ans *= a;
        a *= a;
        p >>= 1;
    }

    return ans;
}

//create reference table (map)
void encoding()
{
    for (int i = ptr - 1; i >= 0; i--) //for (int i = ptr - 1; i >= 0; i--)

```

```

{
    string s = "";
    unsigned char ch = tree[i].data;
    int child = i;
    while (tree[child].parent != -1)
    {
        int parent = tree[child].parent;
        if (tree[parent].leftChild == child)
        {
            s = '0' + s;
        }
        else if (tree[parent].rightChild == child)
        {
            s = '1' + s;
        }
        child = parent;
    }
    mp.insert({ ch, s });
    mp_decoding.insert({ s, ch });
    encoded[cntRef++] = s;
}
}

//create encoded.txt file
void makeCompressedFile(string readPath, string tarPath)
{
    FILE *fout, *fin;
    fout = fopen(readPath.c_str(), "rb");
    fin = fopen(tarPath.c_str(), "w");

    //cout << endl << "Haffman encoding:" << endl;
    while (!feof(fout))
    {
        fputs(mp[fgetc(fout)].c_str(), fin);
        //cout << mp[fgetc(fout)];
    }

    fclose(fout);
    fclose(fin);
    cout << endl << "Successully compressed." << endl;
}

int redundant = 0;
//change the encoding to character
void encodingPress(string cmPath)
{
    string temp_path = "pro.txt";
    int temp = 0;

    FILE* fout, * fin;
    fout = fopen(cmPath.c_str(), "rb");
    fin = fopen(temp_path.c_str(), "w");

    unsigned char ch;
    int cnt = 0;

```

```

int tot = 0;
while (!feof(fout))
{
    temp = (fgetc(fout) - '0') ? temp + fps(2, 7 - cnt) : temp;
    cnt++;

    if (cnt == 8)
    {
        fputc((unsigned char)temp, fin);
        fc += (unsigned char)temp;
        //cout << temp << endl;
        temp = 0;
        cnt = 0;
    }
    tot++;
}
//cout << "*****" << cnt << endl;

if (cnt)
{
    while (cnt != 8)
    {
        redundant++;
        cnt++;
        temp += fps(2, 8 - cnt);
    }

    fputc((unsigned char)temp, fin);
    fc += (unsigned char)temp;
}
//cout << "+++++" << redundant << endl;

fclose(fout);
fclose(fin);
}

//check if the encoding sequence is in the reference table
bool check(string s)
{
    for (int i = 0; i < cntRef; i++)
    {
        if (encoded[i] == s) return true;
    }
    return false;
}

//convert a char to its binary form string
string charToBChar(unsigned char ch)
{
    //cout << "*****" << setbase(16) << ch + 0 << endl;
    string s = "00000000";

    for (int i = 7; i >= 0; i--)

```

```

{
    if (ch & 1) s[i] = '1';
    ch >>= 1;
}
//cout << s << endl;
return s;
}

//reshape encoding file
void makeEncodingFile(string pressed)
{
    //string fc;
    /*ifstream inputStream;
    stringstream strStream;

    inputStream.open(pressed);
    strStream << inputStream.rdbuf();
    inputStream.close();
    fc = strStream.str();

    cout << fc.size() << endl;*/

    /*FILE* fp;
    fp = fopen(pressed.c_str(), "r");

    int cnt = 0;
    while (!feof(fp))
    {
        cnt++;
        fc += fgetc(fp);
    }
    cout << "*****" << cnt << endl;*/

    /*fstream fin;
    fin.open(pressed);
    istreambuf_iterator<char> beg(fin), end;

    string fc(beg, end);
    fin.close();*/

    //cout << fc << " * *****" << endl;
    //cout << fc.size() << endl;

    fstream tar;
    tar.open("encoding.txt", ios::out);

    string temp = "";

    for (int i = 0; i < fc.size(); i++)
    {
        temp += charTobChar(fc[i]);

        //cout << charTobChar(fc[i]) << " ";
    }
}

```

```

        cout << endl;

        temp = temp.substr(0, temp.length() - redundant - 1);

        //cout << temp << endl;

        //cout << temp << " *****" << endl;

        tar << temp;
        tar.close();

    }

    //Haffman decoding
    void decoding(string readPath, string tarPath)
    {
        FILE* fout, * fin;
        fout = fopen(readPath.c_str(), "rb");
        fin = fopen(tarPath.c_str(), "wb");

        string temp = "";
        while (!feof(fout))
        {
            temp += fgetc(fout);
            if (check(temp))
            {
                fputc(mp_decoding[temp], fin);
                temp = "";
            }

        }

        fclose(fout);
        fclose(fin);
    }

    //draw file extension from the given path
    string drawExtension(string path)
    {
        string temp = "";
        for (int i = path.size() - 1; i >= 0; i--)
        {
            if (path[i] == '.')
            {
                temp = '.' + temp;
                break;
            }
            else temp = path[i] + temp;
        }

        return temp;
    }

```

```
int main()
{
    string path;
    cout << "Please enter the file to Compress:" << endl; /*test.txt*/
    cin >> path;

    readFile(path);

    placeTreeLeaves();

    setUpHaffmanTree();

    cout << endl;

    for (int i = 0; i < 2 * ptr - 1; i++)
    {
        cout << i << " " << tree[i].weight << " " << tree[i].parent << " " <<
tree[i].leftChild << " " << tree[i].rightChild << endl;
    }

    cout << endl;

    encoding();

    string cmPath = "encoded.txt";
    //cout << "Please enter the path of encoded file:" << endl; /*encoded.txt*/
    //cin >> cmPath;

    makeCompressedFile(path, cmPath);

    encodingPress(cmPath);

    for (int i = 0; i < ptr; i++)
    {
        cout << tree[i].data + 0 << " " << mp[tree[i].data] << endl;
    }

    makeEncodingFile("pro.txt"); //changes \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\

    int choice = 0;
    cout << "decode the compressed file?" << endl << "1 : YES" << endl << "0 : NO" << endl;
    cin >> choice;
    if (choice)
    {
        string decPath = "decoded" + drawExtension(path);
        decoding("encoding.txt", decPath);
    }

    return 0;
}
```

```

/*TEST CODES*/

//check Haffman tree
/*
for (int i = 0; i < 2 * ptr - 1; i++)
{
    cout << i << " " << tree[i].weight << " " << tree[i].parent << " " <<
tree[i].leftChild << " " << tree[i].rightChild << endl;
}

    cout << endl;

*/

//check encoding mapping
/*
for (int i = 0; i < ptr; i++)
{
    cout << tree[i].data + 0 << " " << mp[tree[i].data] << endl;
}

*/

//check reference table
/*
for (int i = 0; i < cntRef; i++)
{
    cout << encoded[i] << endl;
}

*/

```