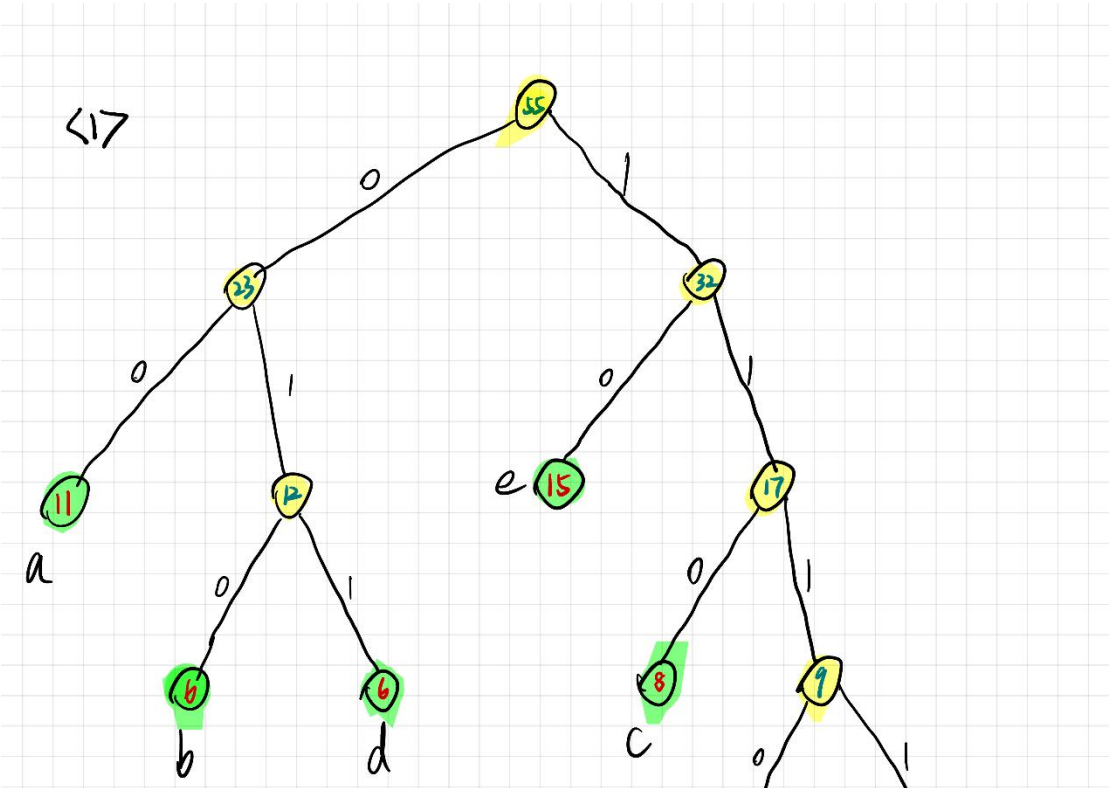
中期文档

21302010042
侯斌洋



哈夫曼编码：

a	00
b	010
c	110
d	011
e	10
f	11110
g	1110
h	11111

注：关于本次 PJ 的代码可以访问以下网站
https://github.com/HBY-STAR/Code/tree/master/DataStructure/project/HuffmanZip
https://gitee.com/hby_star/Code/tree/master/DataStructure/project/HuffmanZip

（2）首先遍历一遍文件，找出每个字符出现的频率并将其保存在一个优先队列（按频率大小排序）中，之后每次取出队列中的频率最小的两个元素并增加一个根节点构造一个子树，根节点的频率为两个最小节点的频率之和，然后再让根节点入队。不断执行上述操作直至队列中只有一个节点，即构造了以此节点为根节点的哈夫曼树，具体实现可参考以下代码：
获取字符频率

```
priority_queue<HuffmanNode, vector<HuffmanNode>, greater<HuffmanNode>>
GetChFreq(const fs::path &file_path, long *file_size)
{
    ifstream input;
    input.open(file_path, ios::in | ios::binary);

    long array[MaxCharNum] = {0};
    long size = 0;
    unsigned char ch;
    HuffmanNode temp;
    priority_queue<HuffmanNode, vector<HuffmanNode>, greater<HuffmanNode>> p_queue;

    while (1)
    {
        input.read((char *)&ch, sizeof(ch));
        if (input.eof())
        {
            break;
        }
        array[ch]++;
        size++;
    }
    *file_size = size;
    for (unsigned int i = 0; i < MaxCharNum; i++)
    {
        if (array[i] != 0)
        {
            temp.ch = i;
            temp.num = array[i];
            temp.Isleaf = true;
            temp.Lnode = nullptr;
            temp.Rnode = nullptr;
            p_queue.push(temp);
        }
    }
    input.close();
    return p_queue;
}
```

构造哈夫曼树

```
HuffmanTree::HuffmanTree(priority_queue<HuffmanNode, vector<HuffmanNode>, greater<HuffmanNode>> &queue)
{
    LeafNum = queue.size();
    if (LeafNum == 0)
    {
        root = nullptr;
    }
    else if (LeafNum == 1)
    {
        root = new HuffmanNode(queue.top());
        queue.pop();
    }
    else
    {
        HuffmanNode *temp, *temp1, *temp2;
        while (queue.size() >= 2)
        {
            temp1 = new HuffmanNode(queue.top());
            queue.pop();
            temp2 = new HuffmanNode(queue.top());
            queue.pop();
            temp = new HuffmanNode(0, temp1->num + temp2->num, false, temp1, temp2);
            queue.push(*temp);
        }
        root = temp;
    }
}
```

（3）如上题中描述使用一个优先队列，这样插入和取出最小平均运行时间都能达到 O(logN)，同时 STL 也提供了优先队列模板方便使用。

（4）对于本次 PJ 我并没有将哈夫曼树存储到文件中，而是存储了文件的字符频率，然后在解压的时候通过读取字符频率再构建哈夫曼树进行解压操作，由于压缩和解压的时候构造哈夫曼树用的是相同的方法，故构造出的哈夫曼树总是一样的。此外只存储字符频率相当于只存储了哈夫曼树的叶节点，故理论上更省空间。

对于本题应该是将哈夫曼树进行序列化之后存储，虽然我没有实现哈夫曼树的序列化，但在下面的文件夹压缩中我对文件夹的树形结构进行了序列化与反序列化，也即 N 叉树的序列化，主要思路就是递归遍历文件夹，具体实现可以参考下面（5）中的文件夹名称的树形结构序列化的附图。

（5）由于之后还要在不解压的情况下进行压缩包预览，故要压缩时要先把文件夹名字的树形结构序列化然后存储在压缩包文件的靠前的位置，在解压或者预览压缩包时便可先读取压缩包文件前面的信息然后反序列化得到文件夹的树形结构，主要思路为递归遍历文件夹，然后对文件和文件夹名称进行写入（序列化）或读取（反序列化）。
具体实现可参考以下代码：

文件夹树形结构序列化的递归函数代码

```
void inFolderCompressTravelDir(const fs::path &folder_path, ofstream &output)
{
    static int file_str_bytes = 0;
    static string tempstr = "";
    static int folder_tag = 0;
    static int right = -1;

    folder_tag = 1;
    tempstr = folder_path.filename().string();
    file_str_bytes = tempstr.size();
    output.write((char *)&folder_tag, sizeof(file_str_bytes));
    output.write((char *)&file_str_bytes, sizeof(file_str_bytes));
    output << tempstr;

    for (const fs::directory_entry &entry : fs::directory_iterator(folder_path))
    {
        if (entry.is_directory())
        {
            inFolderCompressTravelDir(entry.path(), output);
        }
        else
        {
            folder_tag = 0;
            tempstr = entry.path().filename().string();
            file_str_bytes = tempstr.size();
            output.write((char *)&folder_tag, sizeof(file_str_bytes));
            output.write((char *)&file_str_bytes, sizeof(file_str_bytes));
            output << tempstr;
        }
    }
    output.write((char *)&right, sizeof(right));
}
```

文件夹树形结构反序列化的递归函数代码

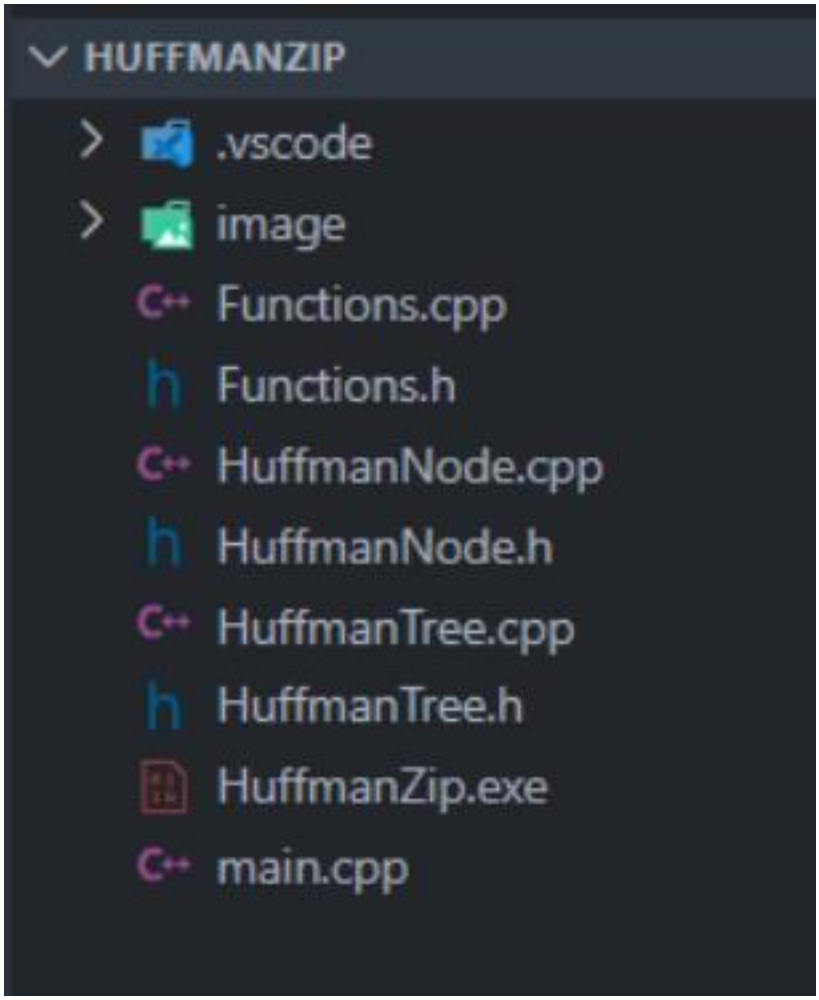
```
void inFolderUncompressTravelDir(ifstream &input, const fs::path &folder_path)
{
    static int file_str_bytes = 0;
    static unsigned char tempch = 0;
    static int folder_tag = 0;

    input.read((char *)&folder_tag, sizeof(folder_tag));

    while (folder_tag != -1)
    {
        if (folder_tag == 0)
        {
            string file_str;
            input.read((char *)&file_str_bytes, sizeof(file_str_bytes));
            for (int i = 0; i < file_str_bytes; i++)
            {
                input.read((char *)&tempch, sizeof(tempch));
                file_str += tempch;
            }
            fs::path new_file(folder_path / file_str);
            fstream file(new_file, ios::out | ios::trunc);
            file.close();
        }
        else if (folder_tag == 1)
        {
            string folder_str;
            input.read((char *)&file_str_bytes, sizeof(file_str_bytes));
            for (int i = 0; i < file_str_bytes; i++)
            {
                input.read((char *)&tempch, sizeof(tempch));
                folder_str += tempch;
            }
            fs::path new_folder(folder_path / folder_str);
            fs::create_directory(new_folder);
            inFolderUncompressTravelDir(input, new_folder);
        }
        input.read((char *)&folder_tag, sizeof(folder_tag));
    }
}
```

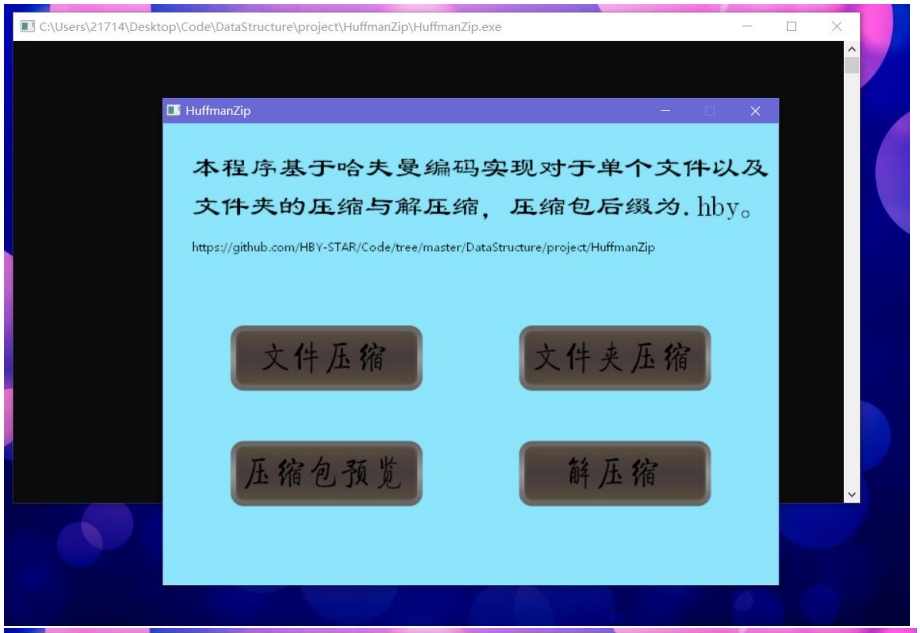
（6）在写中期文档之前整个项目基本上已经完成，主要功能已实现，此外还利用 EGE 图形库简单做了一个界面。
关于本次 PJ 的代码可以访问以下网站
https://github.com/HBY-STAR/Code/tree/master/DataStructure/project/HuffmanZip
https://gitee.com/hby_star/Code/tree/master/DataStructure/project/HuffmanZip

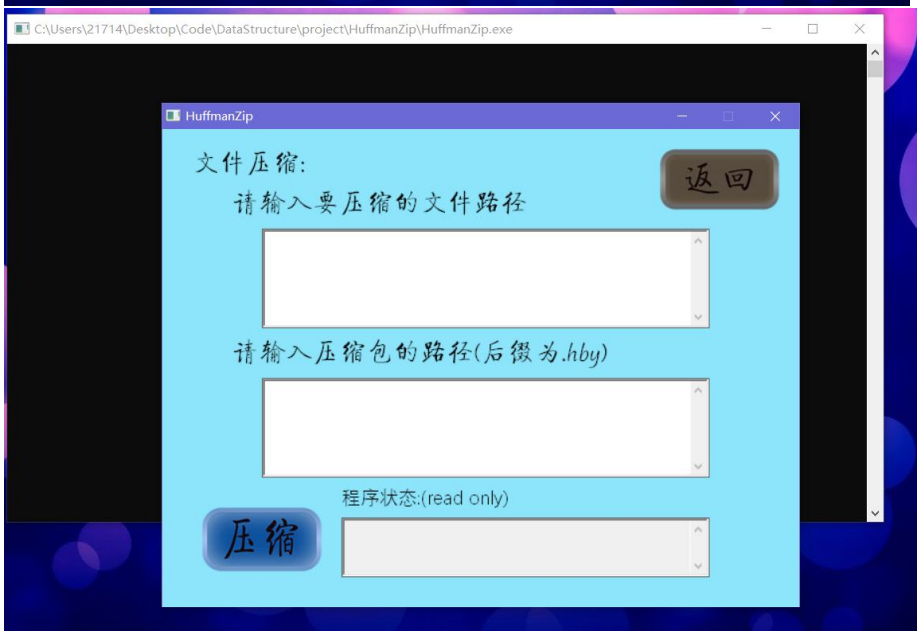
项目文件夹如下：

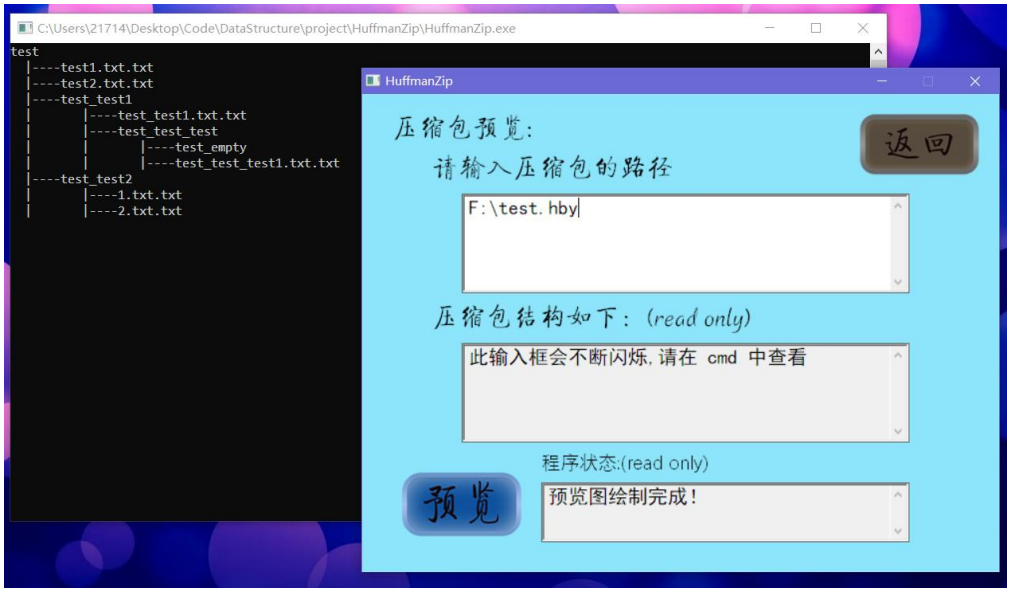


其中：①.vscode 文件夹存储 vscode 的配置文件。
②image 文件夹存储图形界面中要使用的一些图片文件。
③main.cpp 为主程序，里面有项目文件夹中唯一的 main 函数。
④Functions.h 和 Functions.cpp 文件存储 main.cpp 中要用到的函数的声明和实现。
⑤HuffmanNode.h 和 HuffmanNode.cpp 文件存储哈夫曼树节点的类及其成员函数的声明与实现。
⑥HuffmanTree.h 和 HuffmanTree.cpp 文件存储哈夫曼树的类及其成员函数的声明与实现。
⑦HuffmanZip.exe 文件为可执行文件，将其与 image 文件夹放于同一目录下便可直接运行，若无 image 文件夹则图形界面会缺少图片。

部分运行界面如下：


C:\Users\21714\Desktop\Code\DataStructure\project\HuffmanZip\HuffmanZip.exe
HuffmanZip
本程序基于哈夫曼编码实现对于单个文件以及文件夹的压缩与解压缩，压缩包后缀为.hby。
https://github.com/HBY-STAR/Code/tree/master/DataStructure/project/HuffmanZip
文件压缩
文件夹压缩
压缩包预览
解压缩


C:\Users\21714\Desktop\Code\DataStructure\project\HuffmanZip\HuffmanZip.exe
HuffmanZip
文件压缩:
返回
请输入要压缩的文件路径
请输入压缩包的路径(后缀为.hby)
程序状态:(read only)
压缩


C:\Users\21714\Desktop\Code\DataStructure\project\HuffmanZip\HuffmanZip.exe
test
|----test1.txt.txt
|----test2.txt.txt
|----test_test1
| |----test_test1.txt.txt
| |----test_test_test
| | |----test_empty
| | |----test_test_test1.txt.txt
|----test_test2
| |----1.txt.txt
| |----2.txt.txt
HuffmanZip
压缩包预览:
返回
请输入压缩包的路径
F:\test.hby
压缩包结构如下：(read only)
此输入框会不断闪烁，请在 cmd 中查看
程序状态:(read only)
预览
预览图绘制完成！