**系统实现**

# 技术选型

**使用C++语言，基于哈夫曼树原理设计的文件压缩与解压器**

# 系统实现

## 2.1功能选择

该模块通过获取用户输入执行不同函数，实现用户简便的选择压缩或解压缩功能。

在执行程序后，会看到程序的介绍和选择菜单：

**This program uses the Huffman coding algorithm for compression.**

**Any type of file can be encoded using a Huffman code.**

**Decompressing the result will faithfully reproduce the original.**

**Your options are:**

**C) compress file**

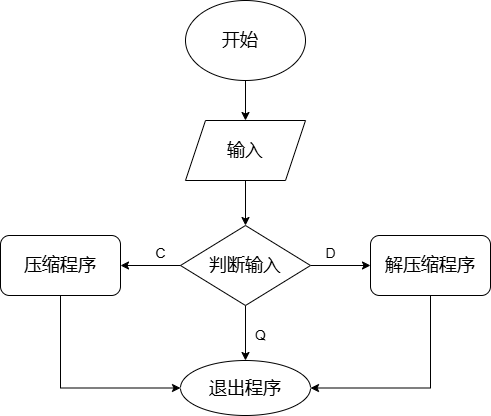
**D) decompress file**

**Q) quit**

**Enter your choice:**

用户在程序中输入“C”开始使用压缩功能，输入“D”开始使用解压缩功能，输入“Q”则可以直接退出程序。

功能选择流程图如下



## 2.2压缩程序

该模块通过获取用户输入的文件名进行文件的压缩。当系统提示输入文件名时，需要指定相对于项目文件夹的名称：

**Input file name:**

**Output file name (Enter for example.txt.huf):**

程序会读取目标文件大小并执行压缩：

**Reading XXX input bytes.**

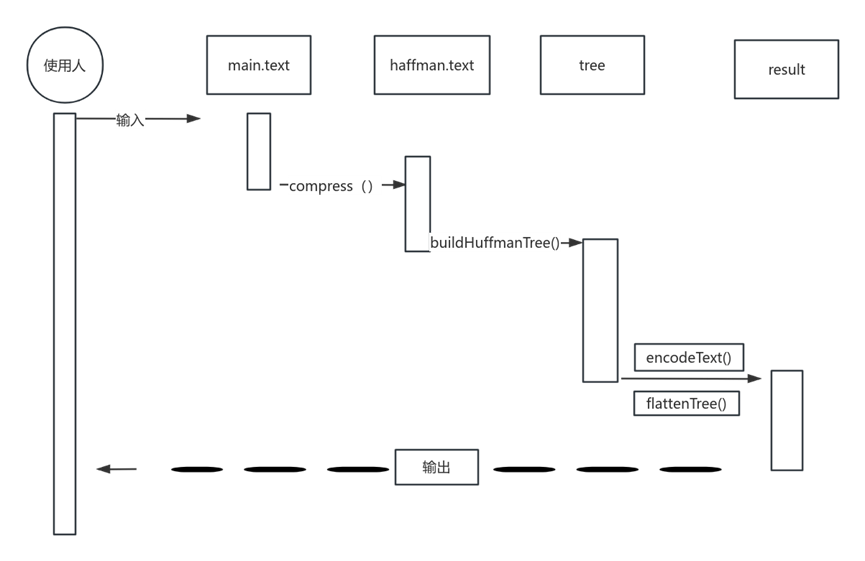
**Compressing ...**

压缩版本将被写入以原文件名和.huf为后缀命名的文件，用户也可以输入其他文件名来修改压缩后文件名，执行结束后会输出压缩后文件大小，可在对应文件夹下找到压缩后文件：

**Wrote XXX compressed bytes.**

模块通过函数Queue encodeText(EncodingTreeNode\* tree, string text)和函数

void flattenTree(EncodingTreeNode\* tree, Queue& treeShape, Queue& treeLeaves)将文件输出为Queue（treeshape）和 Queue（treeleaves）两个返回值并传输给函数EncodingTreeNode\* buildHuffmanTree(string text) buildHuffmanTree。

 EncodingTreeNode\* buildHuffmanTree(string text) buildHuffmanTree 是整个程序的 核心。它接受输入文本并遵循 huffman 算法构建一个针对输入文本的最优 huffman 编码树。随后将编码树传递给函数EncodedData compress(string messageText) 进行压缩。  
 流程图示如下

核心代码如下

EncodingTreeNode\* buildHuffmanTree(string text) {

char prev = text[0];

bool flag = true;

for(char c : text){

if(prev != c){

flag = false;

break;

}

prev = c;

}

if(flag){

error("The message text does not contain at least two distinct characters");

}

Map<char,int> frequency;

for(char i : text){

frequency[i]++;

}

PriorityQueue<EncodingTreeNode\*> pq;

for(char i : text){

if(frequency.containsKey(i)){

EncodingTreeNode\* node = new EncodingTreeNode(i);

pq.enqueue(node,frequency.get(i));

frequency.remove(i);

}

}

while(pq.size() != 1){

int priority = pq.peekPriority();

EncodingTreeNode\* one = pq.dequeue();

priority += pq.peekPriority();

EncodingTreeNode\* zero = pq.dequeue();

EncodingTreeNode\* newNode = new EncodingTreeNode(one,zero);

pq.enqueue(newNode,priority);

}

return pq.dequeue();

}

## 2.3解压缩程序

与文件压缩相似，该模块通过获取用户输入的文件名进行文件的解压缩。当提示你输入文件名时，指定相对于项目文件夹的名称：

**Input file name:**

**Output file name (Enter for unhuf.example):**

程序会读取目标文件大小并执行压缩：

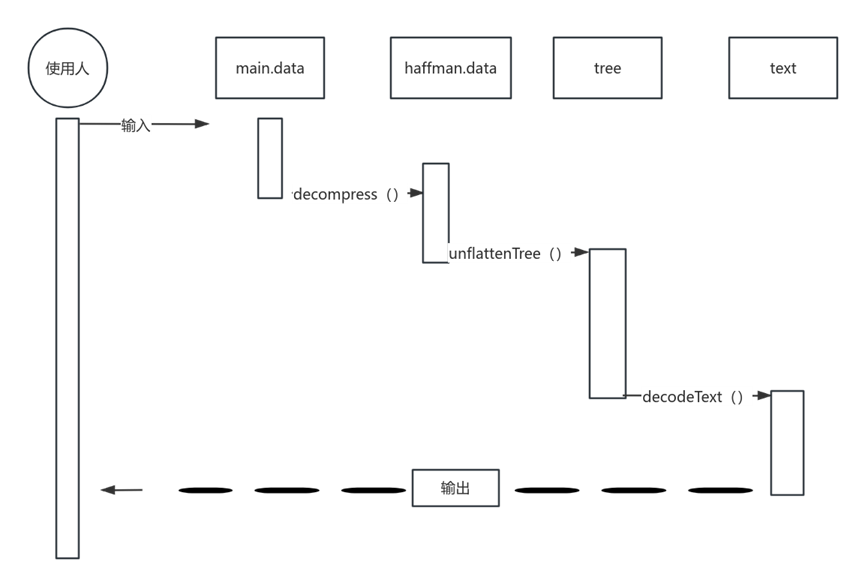
**Reading XXX input bytes.**

**Decompressing ...**

解压缩版本将被写入以原文件名和unhuf.为前缀命名的文件，用户也可以输入其他文件名来修改解压缩文件名，执行结束后会输出解压缩文件大小，可在对应文件夹下找到压缩后文件：

**Wrote XXX compressed bytes.**

该模块通过string decodeText(EncodingTreeNode\* tree, Queue<Bit>& messageBits)函数实现编码后文件转化为解码成原本的编码树信息，再通过函数EncodingTreeNode\* unflattenTree(Queue<Bit>& treeShape, Queue<char>& treeLeaves)利用以上信息进行编码树的重构，最后利用string decompress(EncodedData& data)函数还原原本的信息并写入解压缩后的文件里。

流程图示如下

核心代码如下

void decompressFile() {

string inFilename, outFilename;

if (!getInputAndOutputFiles(inFilename, outFilename, false)) {

return;

}

cout << "Reading " << fileSize(inFilename) << " input bytes." << endl;

try {

ifstream input(inFilename, ios::binary);

EncodedData data = readData(input);

cout << "Decompressing ..." << endl;

string text = decompress(data);

writeEntireBinaryFile(outFilename, text);

} catch (ErrorException& e) {

cout << "Ooops! " << e.getMessage() << endl;

}

if (fileExists(outFilename)) {

cout << "Wrote " << fileSize(outFilename) << " decompressed bytes." << endl;

} else {

cout << "Decompressed output file was not found; perhaps there was an error." << endl;

}

}