

# 中期文档

---

## 问题解答

---

**问题 1: 哈夫曼编码算法是否总能保证最优压缩？如果不一定，请举例说明在哪些情况下它可能不是最优的。**

哈夫曼编码算法在大多数情况下能保证最优压缩，但在某些情况下可能不是最优的。例如，当字符频率分布非常不均匀时，哈夫曼编码可能不是最优的。

**问题 2: 如何根据文件的字节流，构建哈夫曼树？**

- 1. 统计每个字节的出现频率。
- 2. 创建一个优先队列，将每个字节作为一个节点插入队列，优先级为其频率。
- 3. 从队列中取出两个频率最低的节点，合并成一个新节点，新节点的频率为两个节点频率之和。
- 4. 将新节点插入队列。
- 5. 重复步骤 3 和 4，直到队列中只剩一个节点，该节点即为哈夫曼树的根节点。

**问题 3: 对于一组频率已经给定的字符，如果哈夫曼树已经构建完成，是否能快速找到一个新的字符的编码，而不重新构建整棵树？如何实现？**

可以通过调整现有的哈夫曼树来快速找到新字符的编码。具体实现方法是：

- 1. 将新字符作为一个新节点插入优先队列。
- 2. 从队列中取出两个频率最低的节点，合并成一个新节点。
- 3. 重复上述步骤，直到队列中只剩一个节点。

**问题 4: 假设对一个文件进行了哈夫曼编码压缩，如何通过编码表和压缩后的字节流，准确还原出原始文件内容？**

- 1. 根据编码表构建哈夫曼树。
- 2. 从压缩后的字节流中逐位读取数据，通过哈夫曼树进行解码。
- 3. 遇到叶子节点时，输出对应的字符。
- 4. 重复上述步骤，直到读取完所有数据。

**问题 5: 构建哈夫曼树的过程中，如何每次高效、便捷地选出出现频率最低的两个节点？**

可以使用优先队列（最小堆）来高效地选出频率最低的两个节点。插入和删除操作的时间复杂度均为  $O(\log n)$ 。

**问题 6: 如何完成文件夹的压缩并保留内部文件名等信息的一致性？**

- 1. 遍历文件夹，记录每个文件的路径和内容。
- 2. 对每个文件进行哈夫曼编码压缩，并记录每个文件压缩的字节数。
- 3. 将压缩后的数据和文件路径一起存储，形成一个压缩包，并将每个文件的字节数写到压缩文件末尾便于查看。

- 4. 解压时，根据存储的路径信息还原文件夹结构，根据文件末尾的信息来分别读取数据进行还原。

## 问题 7: 如果需要对大量的小文件进行压缩，而不是单个大文件，哈夫曼编码的效率如何？是否有优化的空间？

对大量小文件进行压缩时，哈夫曼编码的效率可能较低。可以通过以下方法优化：

- 合并小文件，把所有的看做一个大文件构建哈夫曼树，然后再压缩。

## 项目进度简述

---

### 已完成的项目需求

- 文件的压缩和解压缩部分全部完成
  - 能够正常压缩/解压给定的一个非空文件
  - 能够正常压缩/解压一个空文件
  - 压缩时，能够指定压缩包的名称；解压时，能够还原出原本的文件名，即便压缩包名称与文件名不同
- 文件夹的压缩和解压缩部分全部完成
  - 能够正常压缩/解压给定的一个非空文件夹
  - 能够正常压缩/解压一个空文件夹
  - 解压时能够还原出原本的文件名、文件夹名
- 此外，我对**压缩解压缩时间**进行了一定程度的**优化**，提高了工作效率

### 未完成的项目需求

- 设置压缩密码
- 文件覆盖问题
- 鲁棒性(只完成了一部分)以及用户交互规则（现在用户根据命令行提示可以完成压缩解压缩操作）

### 接下来的工期计划

- 实现为压缩设置压缩密码
- 解决文件覆盖问题
- 完善鲁棒性并为用户设置使用参数

## 附录

此外主体代码如下：

## 压缩文件代码

```
1 // 压缩单个文件
2 void FileIO::compressFile(const string &filename, const string
&outputFileName)
3 {
4     fs::path entry(filename);
5     // 对于空文件，直接写入0
6     if (std::filesystem::file_size(entry) == 0)
7     {
8         // 读取原文件
9         ifstream inputFile(filename, ios::in | ios::binary);
10        // 以追加模式写入，不知道会不会有影响
11        ofstream outputFile(outputFileName, ios::out | ios::binary |
ios::app);
12
13        // 写入文件头信息
14        fileHead filehead;
15        filehead.originBytes = 0;
16        filehead.alphaVarity = 0;
17        filehead.nameLength = filename.length();
18        strncpy(filehead.name, filename.c_str(), sizeof(filehead.name) -
1); // 将文件名存入字符数组
19
20        outputFile.write(reinterpret_cast<char *>(&filehead),
sizeof(filehead));
21
22        outputFile.write("\n", 1);
23
24        inputFile.close();
25        outputFile.close();
26    }
27    else
28    {
29        // 读取文件并构建字符频率表，在后面可以改到前一层，把这4行作为参数传进来
30        // 这样应该可以解决解压缩文件代码中的错误（好像不太行，这是独立的）
31        map<char, long long> freqTable = makeCharFreq(filename);
32        HuffmanTree tree(freqTable);
33        tree.createHuffmanTree();
34        unordered_map<char, string> charCode = tree.createHuffmanCode();
35
36        // 写入压缩文件
37        ifstream inputFile(filename, ios::in | ios::binary);
38        // 以追加模式写入，不知道会不会有影响
39        ofstream outputFile(outputFileName, ios::out | ios::binary |
ios::app);
40
41        // 写入文件头信息
42        fileHead filehead;
43        filehead.originBytes = inputFile.seekg(0, ios::end).tellg();
44        inputFile.seekg(0, ios::beg);
45        filehead.alphaVarity = charCode.size();
46        filehead.nameLength = filename.length();
47        strncpy(filehead.name, filename.c_str(), sizeof(filehead.name) -
1); // 将文件名存入字符数组
```

```

48     outputFile.write(reinterpret_cast<char *>(&filehead),
sizeof(filehead));
49
50     // 写入字符频度信息
51     for (auto &entry : freqTable)
52     {
53         alphaCode af(entry);
54         outputFile.write(reinterpret_cast<char *>(&af), sizeof(af));
55     }
56
57     // 写入主内容
58     char inputBuffer[BUFFER_SIZE];
59     char outputBuffer[BUFFER_SIZE];
60     int outputIndex = 0;
61     unsigned char bits = 0;
62     int bitcount = 0;
63     long long filesize = fs::file_size(filename);
64     int times = filesize / BUFFER_SIZE;
65     for(int i = 0; i < times; i++){
66         inputFile.read(inputBuffer, BUFFER_SIZE * sizeof(char));
67         for(size_t i = 0; i < BUFFER_SIZE; i++){
68             string currentChar = charCode[inputBuffer[i]];
69             int length = currentChar.length();
70             for(size_t j = 0; j < length; j++){
71                 bits <<= 1;
72                 bits |= (currentChar[j] == '1');
73                 bitcount++;
74                 if(bitcount == 8){
75                     outputBuffer[outputIndex++] = bits;
76                     bits = 0;
77                     bitcount = 0;
78                 }
79                 if(outputIndex == BUFFER_SIZE){
80                     outputFile.write(outputBuffer, outputIndex);
81                     outputIndex = 0;
82                 }
83             }
84         }
85     }
86     // 对于不满BUFFER_SIZE的部分处理
87     long long others = filesize % BUFFER_SIZE;
88     inputFile.read(inputBuffer, others * sizeof(char));
89     for(size_t i = 0; i < others; i++){
90         string currentChar = charCode[inputBuffer[i]];
91         int length = currentChar.length();
92         for(size_t j = 0; j < length; j++){
93             bits <<= 1;
94             bits |= (currentChar[j] == '1');
95             bitcount++;
96             if(bitcount == 8){
97                 outputBuffer[outputIndex++] = bits;
98                 bits = 0;
99                 bitcount = 0;
100             }
101             if(outputIndex == BUFFER_SIZE){
102                 outputFile.write(outputBuffer, outputIndex);

```

```

103         outputIndex = 0;
104     }
105 }
106 }
107 if (!!bitcount)
108 {
109     // 补齐八位
110     for(int i = bitcount; i < 8;i++){
111         bits <<= 1;
112         bits |= 0;
113     }
114     outputBuffer[outputIndex++] = bits;
115 }
116 outputFile.write(outputBuffer, outputIndex);
117 inputFile.close();
118 outputFile.close();
119 }
120 }

```

## 解压缩文件代码

```

1 // 解压缩文件
2 streampos FileIO::decompressFile(const string &filename, string
&outputFileName, long long filesize,streampos startIndex)
3 {
4     // 读取头文件信息
5     auto [filehead,currentPos] = readFileHead(filename,startIndex);
6
7     // 恢复文件名,将输出路径更新
8     string outputFilename(filehead.name, filehead.nameLength);
9     outputFileName = outputFilename;
10    if(filehead.originBytes == 0){
11        ofstream outputFile(outputFileName, ios::out | ios::binary
|ios::app);
12        outputFile.close();
13        return currentPos;
14    }
15    // 读取字符频度信息
16    auto [freqTable, newPos] = readCompressTFileFreq(filename,
filehead.alphaVarity,currentPos);
17
18    // 构建哈夫曼树
19    HuffmanTree tree(freqTable);
20    tree.createHuffmanTree();
21    // 返回子节点
22    HuffmanNode *root = tree.getHuffmanRoot();
23    HuffmanNode *current = root;
24
25    ifstream inputFile(filename, ios::in | ios::binary);
26    // 这个可能也要处理
27    ofstream outputFile(outputFileName, ios::out | ios::binary |ios::app);
28    // 定位到存储文件的位置
29    inputFile.seekg(newPos);

```

```

30 // 缓冲区
31 char inputBuffer[BUFFER_SIZE];
32 char outputBuffer[BUFFER_SIZE];
33 int outputIndex = 0;
34 int writeByte = 0;
35 // 主题内容的字节数
36 long long mainSize = fileSize - (newPos - startIndex);
37 int times = mainSize / BUFFER_SIZE;
38 long long others = mainSize % BUFFER_SIZE;
39 for(int i = 0; i < times; i++) {
40     inputFile.read(inputBuffer, BUFFER_SIZE * sizeof(char));
41     for (size_t i = 0; i < BUFFER_SIZE; i++)
42     {
43         char byte = inputBuffer[i];
44         for (int j = 0; j < 8; j++)
45         {
46             bool bit = byte & (1 << (7 - j));
47             current = bit ? current->right : current->left;
48             if (!current->left && !current->right)
49             {
50                 outputBuffer[outputIndex++] = current->data;
51                 writeByte++;
52                 current = root;
53                 if(outputIndex == BUFFER_SIZE ){
54                     outputFile.write(outputBuffer, outputIndex);
55                     outputIndex = 0;
56                 }
57                 // 处理多余的字符
58                 if(writeByte >= filehead.originBytes){
59                     goto finish;
60                 }
61             }
62         }
63     }
64 }
65 // 对于不满BUFFER_SIZE的部分
66 inputFile.read(inputBuffer, others * sizeof(char));
67 for (size_t i = 0; i < others; i++)
68 {
69     char byte = inputBuffer[i];
70     for (int j = 0; j < 8; j++)
71     {
72         bool bit = byte & (1 << (7 - j));
73         current = bit ? current->right : current->left;
74         if (!current->left && !current->right)
75         {
76             outputBuffer[outputIndex++] = current->data;
77             writeByte++;
78             current = root;
79             if(outputIndex == BUFFER_SIZE ){
80                 outputFile.write(outputBuffer, outputIndex);
81                 outputIndex = 0;
82             }
83             // 处理多余的字符
84             if(writeByte >= filehead.originBytes){
85                 goto finish;

```

```

86         }
87     }
88 }
89 }
90 finish: ;
91     if(outputIndex > 0){
92         outputFile.write(outputBuffer, outputIndex);
93     }
94     streampos nowPos = inputFile.tellg();
95     // 关闭文件
96     inputFile.close();
97     return nowPos;
98 }

```

## 压缩文件夹代码

```

1  // 压缩文件夹
2  void Features::compressDirectory(const string &dirPath, const string
&outputFileName)
3  {
4      // 获取目录下的文件夹信息以及文件信息
5      vector<string> dirname;
6      vector<string> filename;
7
8      // 将原文件夹路径完整的记录下来
9      dirname.push_back(dirPath);
10     // 遍历这个dirPath文件夹,将文件加入到filename,将文件夹加入到dirname,相对路径了
11     // 这里存储的是 /什么什么 注意"/"
12     for (const auto &entry : fs::recursive_directory_iterator(dirPath))
13     {
14         if (fs::is_directory(entry.path()))
15         {
16             dirname.push_back(entry.path().string());
17         }
18         else if (fs::is_regular_file(entry.path()))
19         {
20             filename.push_back(entry.path().string());
21         }
22     }
23
24     // 打开文件开写
25     ofstream output(outputFileName, ios::out | ios::app | ios::binary);
26     // 写入目录名长度
27     int dirnameSize = dirname.size();
28     output.write(reinterpret_cast<char *>(&dirnameSize),
sizeof(dirnameSize));
29
30     // 写入目录名内容
31     for (const auto &dir : dirname)
32     {
33         int dirLength = dir.size();
34         output.write(reinterpret_cast<char *>(&dirLength),
sizeof(dirLength));
35         output.write(dir.c_str(), dirLength);
36     }

```

```

37
38     // 写入文件名长度
39     int filenameSize = filename.size();
40     output.write(reinterpret_cast<char *>(&filenameSize),
sizeof(filenameSize));
41
42     // 写入文件名内容
43     for (const auto &file : filename)
44     {
45         int fileLength = file.size();
46         output.write(reinterpret_cast<char *>(&fileLength),
sizeof(fileLength));
47         output.write(file.c_str(), fileLength);
48     }
49     output.close();
50
51     // 记录每个压缩文件的大小
52     long long filesize[filenameSize];
53     // 挨个压缩文件
54     int i = 0;
55     for (const auto &file : filename)
56     {
57         long long size = compressFile(file, outputFileName);
58         filesize[i++] = size;
59     }
60
61     // 写压缩后的大小,文本文件形式打开
62     ofstream output_1(outputFileName, ios::out | ios::app);
63     output_1 << "\n";
64     for (int i = 0; i < filenameSize; i++)
65     {
66         output_1 << filesize[i] << " ";
67     }
68 }

```

## 解压缩文件代码

```

1 void Features::decompressDir(const string &filename)
2 {
3     ifstream inputFile(filename, ios::in | ios::binary);
4     int dirnameSize, filenameSize;
5     string path;
6
7     // 读取目录名长度
8     inputFile.read(reinterpret_cast<char *>(&dirnameSize),
sizeof(dirnameSize));
9
10    // 读取目录名内容
11    for (int i = 0; i < dirnameSize; i++)
12    {
13        int pathLength;
14        inputFile.read(reinterpret_cast<char *>(&pathLength),
sizeof(pathLength));
15        path.resize(pathLength);
16        inputFile.read(&path[0], pathLength);

```



```
17     fs::create_directories(path); // 创建目录
18 }
19
20 // 读取文件名长度
21 inputFile.read(reinterpret_cast<char *>(&filenameSize),
sizeof(filenameSize));
22
23 // 读取文件名内容
24 vector<string> filepath;
25 filepath.reserve(filenameSize);
26 for (int i = 0; i < filenameSize; ++i)
27 {
28     int fileLength;
29     inputFile.read(reinterpret_cast<char *>(&fileLength),
sizeof(fileLength));
30     path.resize(fileLength);
31     inputFile.read(&path[0], fileLength);
32     filepath.push_back(path);
33 }
34 streampos startIndex = inputFile.tellg();
35 inputFile.close();
36 // 获得每个文件的压缩文件的大小
37 long long *filesize = getCompressDirSize(filename, filenameSize);
38 // 对各个文件进行解压
39 for (int i = 0; i < filenameSize; i++)
40 {
41     FileIO fileIO;
42     startIndex = fileIO.decompressFile(filename, filepath[i],
filesize[i], startIndex);
43 }
44 }
```