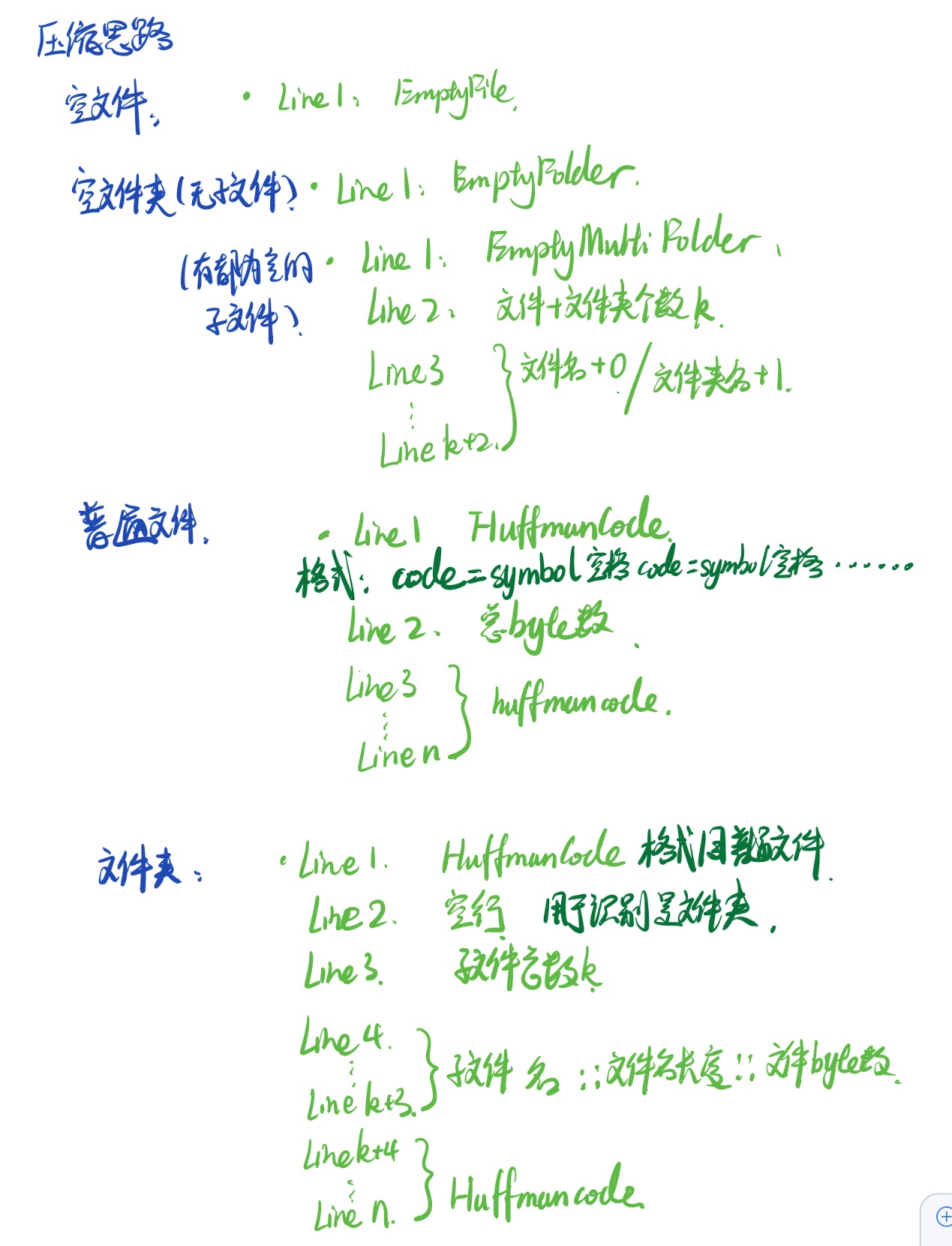
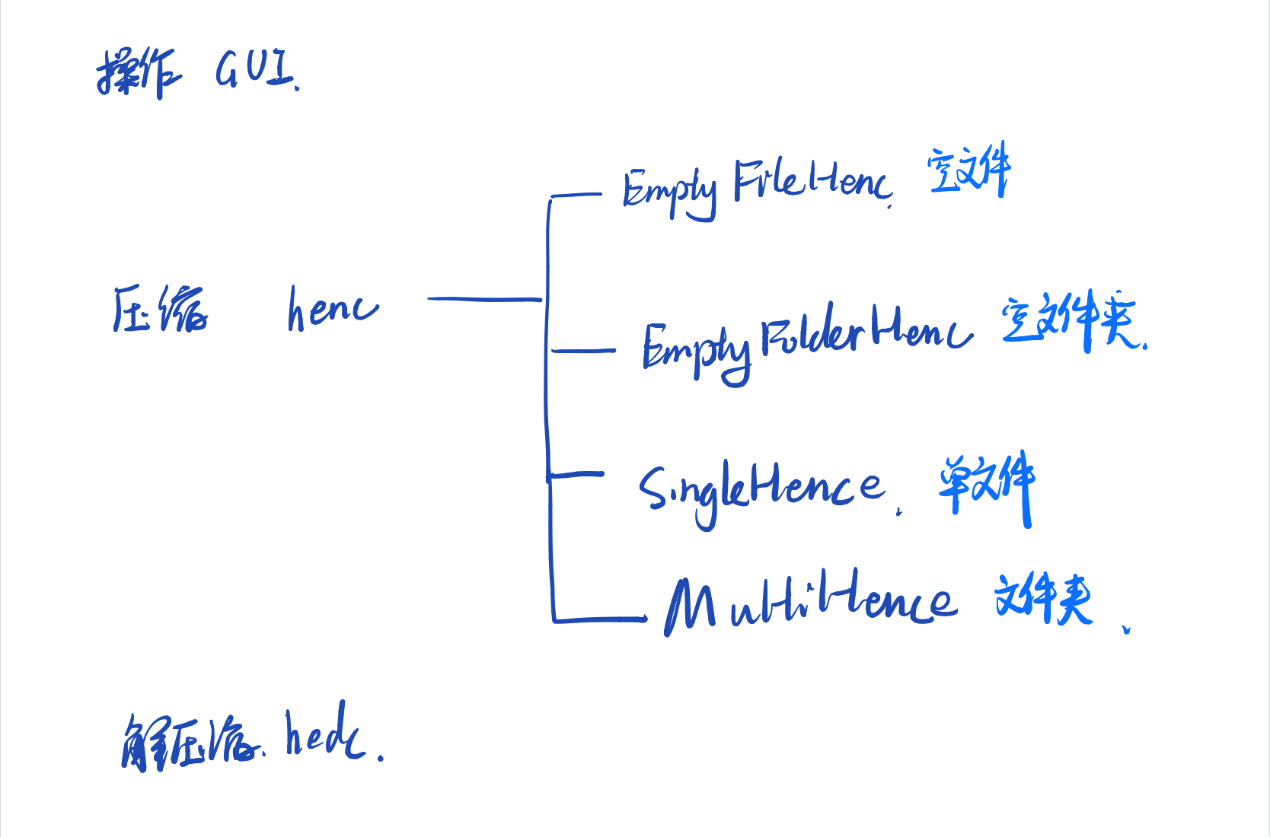
**开发手册**

1. **程序设计构思**



**2.压缩性能**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 压缩前大小 | 压缩后大小 | Zip压缩后大小 |
| **Large\_file/1.jpg** | 20262kb | 20209kb | 20239kb |
| **Largefile/2.csv** | 113865kb | 73046kb | 28626kb |
| **Largefile/3.csv** | 113792kb | 72491kb | 104028kb |
| **Folder/2** | **9.43mb** | 6132kb | 2187kb |
| **Folder/3** | **2.46mb** | **1651kb** | **555kb** |
| **Folder/1** | **3.22mb** | **1959kb** | **555kb** |
|  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **压缩时间** | **解压缩时间** | **Zip压缩时间** | **Zip解压缩时间** |
| **LargeFile/1.jpg** | **5.058s** | **19.25s** | **3s** | **……** |
| **largeFile/2.csv** | **17.56s** | **60.5s** | **30s** | **11s** |
| **largeFile/3.csv** | **15.3s** | **58.61s** | **30s** | **……** |
| **Folder/2** | **2.08s** | **57.9s** | **……** | **8s** |
| **Folder/3** | **0.786s** | **20.3s** | **……** | **……** |
| **Folder/1** | **1.152s** | **18.1s** | **……** | **……** |
|  |  |  |  |  |

**3 遇到的问题**

**已解决的部分**

* **BufferedReader 和 BufferedInputStream的read读取方式不同，前者读int，后者读byte，用BufferedReader读取树后再用BufferedInputStream读取huffman树，后者不知道从哪里开始？**

**方法：**单文件：跳过两行；文件夹，跳过k+3行（k为文件数）。

* **在读取文件夹HuffmanCode时，如何判定第一个文件的代码读取完毕？**

**方法：**存储过程中，在huffmancode之前不知道改写后的长度，只能存储原长；那么让input的读取流持续读取，在翻译的过程中设置指针，达到子文件原长时，改变到下一个输出流。

* **压缩空文件和空文件时，file的长度都为0，解压时如何判断应该改为一个空文件还是一个空文件夹？**

**方法：**压缩过程中，在文件名后存储文件属性；0为文件，1为文件夹。同时，在获取所有子文件的过程中，如果遇到空文件夹，则记录其名字，否则只记录子文件的名字。

* **解压缩文件时，如果当前文件的所处的文件夹没有建立，怎么办？**

**方法：**用split（“符号”）拆分子文件的绝对路径名，获取拆分后的最后一位和前面所有位构成的字符串，用string,mkdirs()建立所有母文件夹，然后用file.creatNewFile()创建当前文件。

**4.我的优化方案**

* **用BufferedReader中的readline方法直接读取Huffman Tree,减少解压时间**
* **用Byte的单位去切分源文件并构成Huffman树，更加高效**
* **在压缩和解压缩的过程中，写入内容是边翻译边写，避免了一次性写完,String过长可能会遇到的Out Of Memory问题。**
* **在压缩过程中，尽早区分文件类型，避免大段无关代码的读取，节省时间**

**5.我的不足**

* **用String类型存Huffman Code，浪费空间，降低效率**
* **从BufferedReader到BufferedInputStream,为了判断从哪开始，使用（b.read()!=’\n’）k+3次**
* **GUI过于简陋**