开发文档

1. 项目概述

本项目实现了一个基于 **霍夫曼编码** (Huffman Coding) 算法的文件压缩和解压工具，支持压缩和解压文件以及文件夹。项目支持命令行界面（CLI）操作，具备文件和文件夹的递归压缩解压、文件加密、错误提示等功能（暂不支持中文字符）。用户可以通过该工具对大文件（>4GB）、空文件以及不同层级的文件夹进行压缩与解压，且支持文件加密和密码验证。

---

2. 项目结构

2.1 目录结构

```
src/
├── CLI/
|   ├── CLI.cpp                // 实现命令行接口类的方法，负责与用户交互，解析输入参数。
|   └── CLI.h                  // 定义了命令行接口类，负责解析用户输入的命令。
├── FILEIO/
|   ├── compress.cpp           // 实现文件夹压缩功能。
|   ├── compress.h             // 声明文件夹压缩功能的接口。
|   ├── decompress.cpp         // 实现文件夹解压缩功能。
|   ├── decompress.h           // 声明文件夹解压缩功能的接口。
|   ├── fileIO_C.cpp           // 实现文件压缩功能。
|   ├── fileIO_C.h             // 声明文件压缩功能的接口。
|   ├── fileIO_D.cpp           // 实现文件解压缩功能。
|   ├── fileIO_D.h             // 声明文件解压缩功能的接口。
|   ├── HuffmanTree.cpp        // 实现霍夫曼树的创建与编码生成过程。
|   └── HuffmanTree.h          // 定义了霍夫曼树和节点类，声明了构建霍夫曼树的方法。
├── UTILS/
|   ├── Utils.cpp              // 包含辅助功能和工具函数的实现。
|   └── Utils.h                // Utils.cpp 的头文件，定义了辅助功能和工具函数。
├── Huffman.exe                // 编译生成的可执行文件。
└── main.cpp                   // 程序入口，调用主要功能。
```

2.2 运行方法

```
 help command:   help                // 输入help打印命令形式
 exit command:   exit                // 输入exit退出程序
 compress command:                   // 压缩指令
    hfm filename outputName           // filename是要压缩的路径,outputName是输出的路径
    hfm filename outputName password  // filename是要压缩的路径,outputName是输出的路径,password是压缩密码
 decompress command:                 // 解压缩指令
    unhfm filename                   // filename是要解压缩的文件路径,默认解压到当前文件夹
    unhfm filename outputName        // filename同上，outputName是输出的文件夹路径
```

2.3 主要类，函数及功能

1. `HuffmanNode` 类

- **功能**：表示霍夫曼树中的一个节点，存储数据字节、频率以及左右子节点的信息。
- 核心方法：
    - `bool operator<(const HuffmanNode &node) const`：实现小顶堆的比较运算符。

2. `HuffmanTree` 类

- **功能**：构建霍夫曼树，提供树的序列化与反序列化功能。
- 核心方法：
    - `void createHuffmanTree()`：构建霍夫曼树。
    - `unordered_map<char, string> createHuffmanCode()`：获取霍夫曼编码。
    - `void subCreateHuffmanCode(HuffmanNode* root, string huffmancode, unordered_map<char, string>& haffmanCode)` ://递归获取哈夫曼编码
    - `HuffmanNode* getHuffmanRoot()`：返回霍夫曼树根节点。

3. `FileIO_C` 类

- **功能**：处理文件的压缩，最低层的内容处理。
- 核心方法：
    - `map<char, long long> makeCharFreq(const string& filename)`：构建字符频率表。
    - `void compressFile(const string& filename, const string& outputFileName, const string &prefix)`：压缩单个文件。
    - `void writeHuffmanTree(ofstream& file, HuffmanNode* root)`：写入哈夫曼树结构。
    - `void compressSmallFile(ifstream &inputFile, ofstream &outputFile, string *charCodeArray, long long filesize)`:正常压缩文件。
    - `void compressLargeFile(ifstream &inputFile, ofstream &outputFile, string *charCodeArray, long long filesize)`:多线程压缩文件。

4. `FileIO_D` 类

- **功能**：处理文件的解压缩，最低层的内容处理。
- 核心方法：
    - `void decompressFile(const string& filename, const string& outputFileName, long long filesize, const streampos &startIndex)`：解压缩单个文件。
    - `pair<fileHead,streampos> readFileHead(const string& filename,const streampos &startIndex)`:读取压缩文件头信息。
    - `HuffmanNode* readHuffmanTree(ifstream& file, long size)`:读取哈夫曼树并返回树的根节点

- `void decompressWithMultiThread(ifstream &inputFile, ofstream &outputFile, HuffmanTree &tree, HuffmanNode *root,long long filesize, streampos newPos, long long originBytes)`: 多线程压缩
- `void decompressWithSingleThread(ifstream &inputFile, ofstream &outputFile,HuffmanTree &tree,  HuffmanNode *root, long long filesize, streampos headSize, long long originBytes)`: 单线程压缩

5. `compress` 类

- **功能**：提供文件夹压缩功能，综合`FileIO_C`**类**来实现例如文件加密等的上层操作。
- 核心方法：
  - `void compress(const string& filename, const string& outputFileName, const string &password)`：压缩文件或文件夹。
  - `long long compressFile(const string& filename, const string& outputFileName, const string &prefix)`: 压缩单个文件
  - `void compressDirectory(const string& dirPath, const string& outputFileName)`: 压缩文件夹

6. `decompress` 类

- **功能**：提供文件夹解压缩功能，综合`FileIO_D`**类**来实现例如文件解密等的上层操作。
- 核心方法：
  - `void decompress(const string& filename, string& outputFileName, int passLength)`: 解压缩文件或文件夹
  - `void decompressFile(const string& filename, string& outputFileName, streampos currentPos)`: 解压缩文件
  - `void decompressDir(const string& filename, const string &prefix, streampos currentPos)`：解压缩文件夹
  - `void decompressFileTask(const string& filename, string& filepath, int filesize, int startIndex)`: 用于多线程解压

7. `CLI` 类

- **功能**：处理命令行接口，与用户交互。
- 核心方法：
  - `void parseArguments(int argc, char* argv[])`：解析命令行参数。
  - `void displayHelp()`：显示帮助信息。

8. `Utils` 函数

- **功能**：提供各种辅助功能和工具函数。
- 核心方法：
  - `int checkOutputPath(const string &filepath)`：检查输出路径是否存在，并提示用户选择操作，用于压缩。

- `long long* getCompressDirSize(const string& filename, int filenameSize)`:获得每个文件的压缩文件的大小
- `int checkOutputPath(const string &filepath)`:检查输出路径是否存在,并提示用户选择操作，用于解压缩
- `int coverAll(const vector<string> &filepath, int filenameSize)`:在相同文件数过多，确认是否一键全部覆盖，还是由用户自己选择，亦或是全部跳过

---

3. 核心需求

3.1 文件的压缩与解压（30%）

需求分析与实现：

- **功能**：支持压缩与解压文件，并保证压缩后的文件内容和原始文件一致。需要支持大文件（>4GB）和空文件的处理。
- **实现思路**：
    - 使用 **霍夫曼编码** 算法进行压缩，压缩后的文件大小大多数小于原始文件。
    - 为避免处理大文件时的 `int` 溢出，所有文件大小计算使用 `long` 或 `long long` 类型。
    - 对空文件的处理，直接将文件头内容的字节数和字符数设置为0，写入文件后直接return。

实现：

- 在 `FileIO` 和 `Features` 类中使用霍夫曼算法压缩和解压文件，确保解压后的数据与原数据一致。
- 采用 `long` 或 `long long` 类型来处理大文件的大小，避免 `int` 类型溢出问题。
- 压缩时可以指定压缩包名称，解压时会自动恢复原文件名及其目录结构。

注：

- **压缩单个文件时的写入顺序如下**：
    1. 先写入密码的长度和大小（没有长度为0）
    2. 使用一个字节来记录是文件，并写入文件的相对路径
    3. 写入文件头信息（字节数，字符种类数量（空文件为0））
    4. 写入哈夫曼树
    5. 用一个字符来判段是多线程压缩还是直接压缩（有的多线程没有直接压缩快）
    6. 然后根据文件的大小分割成多个数据块，每个数据块的对应的哈夫曼压缩的数据按顺序写入压缩文件。

3.2 文件夹的压缩与解压（20%）

需求分析与实现：

- **功能**：支持压缩与解压文件夹，递归处理子文件夹。
- **实现思路**：
    - 使用递归方式遍历文件夹及其子文件夹，压缩所有文件及文件夹结构。
    - 对空文件夹压缩时，仅压缩文件夹结构，不包含文件内容。

实现：

- `Features` 类中的`compressDirectory`通过递归遍历文件夹中的每个文件和子文件夹，调用 `FileIO` 压缩文件。
- `Features` 类中的`decompressDir`读取压缩文件的文件夹信息，恢复文件夹的结构。

注：

- **压缩文件夹时的写入顺序如下**：
    1. 先写入密码的长度和大小（没有长度为0）
    2. 使用一个字节来记录是文件夹
    3. 写入所记录的文件夹的数量和内容（相对路径），再写入所记录的文件的数量和名称（相对路径）
    4. 循环进行5,6,7,8的操作，直至写入完毕
    5. 写入文件头信息（字节数，字符种类数量（空文件为0））
    6. 写入哈夫曼树
    7. 用一个字符来判段是多线程压缩还是直接压缩（有的多线程没有直接压缩快）
    8. 然后根据文件的大小分割成多个数据块，每个数据块的对应的哈夫曼压缩的数据按顺序写入压缩文件。
    9. 最后在文件末尾写入每个文件压缩后所占的字节数（以便读取和跳过）

3.3 设置压缩密码（15%）

需求分析与实现：

- **功能**：在压缩时可以选择设置密码，解压时需要输入密码。
- **实现思路**：
    - 在压缩时，用户可以设置密码，密码可以直接编码进文件中。
    - 如果文件是加密压缩的，解压时要求用户输入密码，若密码正确，则解压。

实现：

- 压缩时，直接将密码存储在压缩包头部，解压时，判断是否有密码的输入，如果有要求输入密码（见`CLI::passwordCorrect`）。
- 由于我的压缩文件的哈夫曼树是按位压缩以及一些情况下会压入一些个人的标记符，所以在不知道逻辑的情况下，很难还原哈夫曼树以及总的数据，故而只将密码压入文件即可

3.4 代码风格（5%）

需求分析与实现：

- **目标**：
    - 确保代码具备良好的可读性和可维护性，便于后续扩展和维护。
    - 遵循面向对象设计原则，实现代码的模块化、职责单一和低耦合。
    - 提供清晰的注释，帮助理解代码逻辑，特别是在复杂算法（如霍夫曼编码）和核心功能部分。

解决方案：

- 模块化设计：项目划分为多个功能模块，如压缩、解压、用户交互（CLI）和工具类模块。每个模块职责单一且功能明确。
- 避免代码冗长：复杂逻辑被拆分为多个独立的私有方法。
- 清晰的注释和命名：每段逻辑都提供了详细的注释，解释了为什么这样设计以及每一步的功能。

一点展示：

```
// 解压缩
void Decompress::decompress(const string& filename, string& outputFileName, int passLength){···
// 解压缩文件
void Decompress::decompressFile(const string &filename, string &outputFileName, streampos currentPos)···
// 单个处理任务
void Decompress::decompressFileTask(const string& filename, string& filepath, int filesize, int startIndex) {···
// 解压缩文件夹
void Decompress::decompressDir(const string &filename, const string &prefix ,streampos currentPos)
{
    ifstream inputFile(filename, ios::in | ios::binary);
    inputFile.seekg(c int filenameSize
    int dirnameSize, filenameSize;
    string path;
    // 判断路径前缀
    string fullPathPrefix = prefix.empty() ? "" : prefix + "\\";

    // 读取目录名长度
    inputFile.read(reinterpret_cast<char *>(&dirnameSize), sizeof(dirnameSize));
    // 读取目录名内容

    for (int i = 0; i < dirnameSize; i++)
    {
        int pathLength;
        inputFile.read(reinterpret_cast<char *>(&pathLength), sizeof(pathLength));
        path.resize(pathLength);
        inputFile.read(&path[0], pathLength);
        fs::create_directories(fullPathPrefix + path); // 创建目录
    }

    // 读取文件名长度
    inputFile.read(reinterpret_cast<char *>(&filenameSize), sizeof(filenameSize));

    // 读取文件名内容
    vector<string> filepath;
    filepath.reserve(filenameSize);
```

4. 其他需求

4.1 用户交互（5%）

需求分析与实现：

- **功能**：用户通过命令行输入操作指令，不支持中文路径。
- **实现**：
    - 在 `CLI` 类中，提供命令行方式压缩和解压文件，支持文件和文件夹操作。

4.2 鲁棒性（5%）

需求分析与实现：

- **功能**：捕获用户输入错误，给出清晰的错误提示。
- **实现**：
    - 在文件路径，部分可能会循环较多次数导致用户想停下来等地方使用 `try-catch` 捕获异常，避免程序崩溃。
    - 考虑用户可能的非常规操作，进行错误提示。

4.3 文件覆盖问题（5%）

需求分析与实现：

- **功能**：在输出路径中有重复文件的时候询问用户如何处理。
- **实现**：
    - 在需要写入新文件时，先对现有路径进行检测，如果发现了重复的文件或文件夹都跳出提示让用户进行选择，根据数量的多少会有不同的提示，用户在重复较多的情况下可以选择一键覆盖，一键跳过，或自己处理。
    - 当选择跳过时，如果是单一文件，直接结束，如果是文件夹类型，会先提前计算这个跳过的文件大小，然后进行下一个文件的处理。

---

5. 开发环境与工具

- **编程语言**：`C++`
- **开发工具**：`Visual Studio Code`
- **版本控制**：`Git`

---

6. 性能测试

测试用例：

测试文件	原始大小	压缩后大小	压缩时长/s	解压时长/s	压缩率
testcase	3,769,450,636 字节	2,565,205,414 字节	37.92	38.47	68.05%

测试文件	原始大小	压缩后大小	压缩时长/s	解压时长/s	压缩率
testcase01EmptyFile\empty.txt	0字节	34 字节	0.00	0.00	nan
testcase02NormalSingleFile	23,903,670 字节	17,492,447 字节	0.54	0.53	73.17%
testcase03XLargeSingleFile	1,105,931,880 字节	709,457,601 字节	3.51	1.96	64.15%
testcase02NormalSingleFile\1.txt	1,985,016 字节	1,110,950 字节	0.06	0.05	55.97%
testcase06SubFolders	448,515,860 字节	447,173,817 字节	25.77	30.23	99.7%
testcase07XlargeSubFolders	1,099,970,162 字节	698,093,682 字节	3.88	2.17	63.46%
testcase08Speed\1.csv	643,412,034 字节	411,715,762 字节	1.85	0.97	63.99%
testcase09Ratio	441,771,600 字节	277,051,286 字节	1.32	0.69	62.71%
testcase03XLargeSingleFile\1.jpg	20,748,246 字节	20,690,468 字节	0.23	0.30	99.72%

---

7. 遇到的问题与解决方案

7.1 频繁IO处理较消耗时间且缓存区内容有限而处理的文件较大?

- 选择BUFFER_SIZE等作为缓冲区的大小，分块进行读取

7.2 如何在解压缩文件夹的时候还原出一个空文件夹?

- 递归遍历文件夹后将路径写入目标文件，解压缩的时候直接读取创建即可

7.3 如何提高压缩和解压缩的速度?

- 在压缩和解压缩的时候加入多线程处理，对于大文件的处理能够加快很多（在文件夹的解压缩时原本准备采用多线程但不知道为什么后面的线程异常结束且没有报错信息，是个迷😨）。
- 在文件较小时缓冲区使用定长数组，较大时看情况使用可变数组（定长数组的处理速度好像比可变数组快）

7.4 使用终端输入用户可能输入非基本命令信息，如何应对?

- 使用命令行参数指定输入输出，避免用户输入错误。
- 对于各种可能出现的情况，进行专业化应对。

7.5 在文件较小时使用多线程没有正常压缩快

- 由于我对正常压缩的时间进行了深入的优化，造成了没有太深入优化的多线程没有正常压缩快
- 所以我采用了两种方式结合的压缩方法，将FILE_SIZE设置为10MB超出的部分采用多线程，小于的部分采用正常压缩加快时间（这里的FILE_SIZE我没有精调）

---

8. 创新点

8.1 还原路径

- 在可以还原原本路径的基础上增加高级功能，用户通过输入目标文件路径即可将文件解压缩或压缩到对应目录下

8.2 极小化哈夫曼树的压缩从而使压缩率尽可能的高

- 采用位压缩的方式，将哈夫曼树信息压缩到最小（0 ->内部节点 , 1->叶子结点，遇到1就压入一个字节的数据，每满一个字节，就存入缓冲区），在计算哈夫曼树所应占的字节时，由于无法先得到，我使用了一点离散数学的知识手动计算哈夫曼树压缩的字节数😄,得出所占字节数然后方便处理。

8.3 加速压缩速度

- 将哈夫曼编码的`unorder_map`转化为`string`数组，减少查找时间（一开始我使用的是`map`结构存储，时间复杂度O(log n)，然后改为`unorder_map`,最后才是数组）
- 通过位运算符来进行1个字节的组装，减少运行时间
- 使用多线程的方法，同时对电脑支持的线程数进行处理，减少运行时间

8.4 加速解压速度

- 采用位运算的方式来进行解压，确保解压速度较大
- 使用多线程的方式进行解压，同时对多个数据进行解压处理

8.5 采用正常压缩和多线程压缩的方式

- 程序实机跑显示我的代码正常压缩部分在文件较小时占优势，文件较大，多线程占优势，故而将两者结合，产生了较快的压缩解压缩方式

9. 结论

本项目实现了一个基于霍夫曼编码的压缩工具，支持命令行操作，能够处理大文件、文件夹及提供加密功能，且运行速度较快。代码设计遵循良好的面向对象设计原则，功能齐全，性能良好，能够满足用户对文件压缩和解压的需求。