项目文件结构说明

```
bin
|---xqz.exe              压缩工具
include              头文件
|---CodeTree.hpp         用于解压的编码树
|---cxxopts.hpp          github上找的命令选项解析器
|---fnode.hpp            用于压缩文件内文件结构的类
|---HuffmanTree.hpp      关于哈夫曼树和哈夫曼编码
|---xq_debug.hpp         debug用
|---xqzio.hpp            关于输入输出的一些函数
src                  源文件
|---CodeTree.cpp
|---fnode.cpp
|---HuffmanTree.cpp
|---xqzio.cpp
|---main.cpp
test
|---performance.xlsx     性能测试结果
.gitignore
Makefile
Project.pdf
README.md              此开发文档
```

项目设计、实现的具体思路

1. 建哈夫曼树&生成哈夫曼编码

- 建哈夫曼树
  - 所有节点放在一个优先队列里
  - 每次取出头部的两个，合并，放回队列，直到队列元素个数为1
- 生成哈夫曼编码
  - 从叶节点往上走，生成编码

```cpp
#define uc unsigned char
typedef std::map<uc, std::string> Word2Code_T;
typedef std::map<std::string, uc> Code2Word_T;
extern int nodes_num; //节点个数
extern Word2Code_T word2code;
extern Code2Word_T code2word;

class HuffmanTreeNode
{
private:
    int value = 0;
    HuffmanTreeNode *father = nullptr;
    char lr = 0; // 0-left_son, 1-right_son
public:
    int idx = 0;
```

```
    HuffmanTreeNode();
    HuffmanTreeNode(int v);
    HuffmanTreeNode(int v, HuffmanTreeNode *real_addr);
    bool operator<(const HuffmanTreeNode &x) const;
    void SetValue(int v);
    void SetFather(HuffmanTreeNode *f);
    void SetLR(char _lr);
    int GetValue();
    char GetLR();
    // GenerateCode - 从此节点向上生成01序列，仅用于叶节点
    std::string GenerateCode();
};

extern std::map<int, HuffmanTreeNode *> idx2nodes;
extern HuffmanTreeNode *word2nodes_addr[256];

struct HuffmanForest_Queue_Cmp{bool operator()(int a, int b);};
typedef std::priority_queue<int, std::vector<int>, HuffmanForest_Queue_Cmp>
HuffmanForest_Queue;

class HuffmanForest
{
private:
    HuffmanForest_Queue nodes;

public:
    HuffmanForest();
    // is_Tree - 是否是一棵树，即是否生成完毕
    bool is_Tree();
    // AddNode - 用于添加初始节点（叶节点）
    void AddNode(HuffmanTreeNode *x, uc word);
    void Pop();
    HuffmanTreeNode *Top();
    void Push(HuffmanTreeNode *x);
    Word2Code_T GenerateWord2Code();
    Code2Word_T GenerateCode2Word();
};

// MergeHuffmanTree - 合并两个节点
HuffmanTreeNode *MergeHuffmanTree(HuffmanTreeNode *x, HuffmanTreeNode *y);
// GenerateHuffmanTree - 生成哈夫曼树
void GenerateHuffmanTree(HuffmanForest *x);
```

2. 压缩

先判断是不是文件

- 文件
  - 文件是否为空
    - 是
      - 按空文件格式写入（格式见下）
    - 否
      - 读入并计数
      - 生成哈夫曼编码

        - 评估压缩后是否有压缩效果
            - 若有，按字节操作，转换成哈夫曼编码写入压缩文件
            - 若无，不压缩，直接copy
- 文件夹
    - 对文件夹内所有文件进行压缩

压缩文件格式

```
[1 or 0]        # 1 - file; 0 - folder
############## if file ################
.\[file_name]
[压缩文件长度(bit)]
# code2word
[code]:[word]
...
*
[compressed file]
############## if folder ##############
.\[folder_name]
[number of empty folder]
\...                    # 空文件夹相对根文件夹路径
...
[number of files]
\[file_path]            # 相对根文件夹路径
[压缩文件长度]           -
# code2word             |
[code]:[word]           |
...                     |
*                       |
[compressed file]       -
...
############ fake compress ###########
.\[file_name]
[文件长度(byte)]
**
...
############## empty file ############
.\[file_name]
0
```

```
extern ull word_freq[256];
extern const int MAX_IO_N;
extern std::vector<std::string> files;
extern std::vector<std::string> empty_folders;

bool isFile(const char* path);
// GetFileSize - 得到文件大小(按byte)
ull GetFileSize(const char* file_path);
// GetFolderSize - 得到文件夹大小(按byte)
ull GetFolderSize(const char* folder_path);
// GetRootPath - 得到根目录
std::string GetRootPath(std::string src);
// GetFolderFiles - 得到root下所有文件，存在files里
void GetFolderFiles(std::string root);
// xqz_read_src_compress - 读文件，并计数得到word_freq
void xqz_read_src_compress(const char *filename);
```

```
// cnt_freq - 对x中字符计数
void cnt_freq(uc *x, int l);
// compressed_length - 压缩文件长度(按bit)
ull compressed_length();
// compressed_code2word_length - 压缩文件加上code2word长度(按byte)
ull compressed_code2word_length();
```

3. 解压

先判断是文件夹还是文件

- 文件
  - 读入文件路径，并创建解压文件所在的文件夹（如果路径不存在）
  - 按照压缩文件格式依次读入各个部分
    - 若为fake compress，直接无需解压，copy到解压文件
    - 否则
      - 建立编码树，就是没有节点频率信息的哈夫曼树
      - 按位操作，0走左儿子，1走右儿子，直到叶节点，于是得到解压出来的word，写入解压文件
- 文件夹
  - 处理空文件夹
  - 处理文件

4. 展示压缩文件结构

- 按照压缩文件格式读入，抛弃除文件路径外的信息
- 按照文件路径创建fnode
- 用printDirTree()输出

性能测试结果

见`./test/performance.xlsx`

思考:

- 哈夫曼压缩对字符出现频率差异较大的文件效果较好，对字符出现频率平均的文件效果很差
- 由于哈夫曼压缩是不定长编码，解压时需要按位操作，解压速度不好提升

遇到的问题和解决方案（项目重难点）

1. mkdir

默认的mkdir函数不能创建多级目录，用递归解决

2. 检查文件或文件夹是否存在

access()

3. 压缩和解压时读写问题

一次性读入0x1000个字符，然后处理

4. 用户输入的路径不规范

```
void parse_path(string &x)
{
    for (int i = 0; i < x.size(); ++i) //所有的'/'换成'\'
    {
        if (x[i] == '/')
        {
            x[i] = '\\';
        }
    }
    if (x[0] != '.' || x[0] != '\\')    //若路径开头没有'.\'，默认为当前目录下
    {
        x = ".\\" + x;
    }
    if (x[x.size() - 1] == '\\')       //路径末尾若有'\'，去掉
        x[x.size() - 1] = '\0';
    return;
}
```

优化技巧（项目亮点）

- 算法
  - 哈夫曼树
    - 利用优先队列生成哈夫曼树
  - 压缩
    - 压缩前评估效果，防止压缩后反而比原文件大很多
  - 解压
    - 解压时生成简单的编码树，节省生成哈夫曼树的时间
    - 解压时用位运算获取每一位的信息
- 功能
  - 多个文件压缩成一个文件

    这种压缩出来的文件，解压时若不用“-n”选项指定文件名，默认会被一个文件夹 `tmp` 包含
  - 展示压缩文件内文件结构
  - 压缩或者解压到指定文件夹
- 用户友好
  - 用命令的形式，就可以自动补全路径
  - 进度条显示
  - help显示

```
xqz - huffman compress by xq
Usage:
  xqz [OPTION...]

  -c, --compress <file/folder>  compress something
  -n, --name <name>             set output name
  -x, --extract <file>          extract .xqz file
  -t, --to <directory>          place the output into <directory>
  -s, --show <file>             show files
  -h, --help                    print help
```

参考

- 获取文件夹下文件目录

  > https://blog.csdn.net/leo_888/article/details/80681184
  >
  > https://blog.csdn.net/sagghab/article/details/81436515 - _finddata_t

- > access( )函数：用于文件读取，判断文件是否存在，并判断文件是否可写。Linux中，函数为access();标准C++中，该函数为_access，两者的使用方法基本相同
  >
  > access(const char *pathname, int mode);//位于<unistd.h>中
  >
  > int _access(const char *pathname, int mode);//位于<io.h>中
  >
  > pathname 为文件路径或目录路径 mode 为访问权限
  >
  > 如果文件具有指定的访问权限，则函数返回0；如果文件不存在或者不能访问指定的权限，则返回-1.
  >
  > mode的值和含义如下所示：
  >
  > 00——只检查文件是否存在
  >
  > 02——写权限
  >
  > 04——读权限
  >
  > 06——读写权限

- 进度条

  > https://www.cnblogs.com/seafever/p/12345200.html

- 命令选项解析器

  > https://github.com/jarro2783/cxxopts