

压缩原理：原本每个字符都需要 8 个 bit 储存，通过 huffman 算法，把出现次数多的字符用更少的 bit 去储存。

需要解决的问题：

- bit 级的读取和写入

- 文件的压缩

- 文件夹的压缩

- 解压

思路：

一．压缩：

读取文件——>统计每个 byte 出现的次数 (weight) ——>构建 huffman 树，获取每个 byte 对应的 Huffman 编码，存入 `hashMap<Byte,String>`

写入压缩文件

1.文件头

存文件时，先存入 `fileType`，以 0 代表该文件是一个文件，1 代表文件夹

对于文件夹，存入 `fileType=1`，存入文件夹里包含的文件个数，存入文件名的长度，再存入 String 文件名

对于文件，存入 `fileType=0`，存入文件名的长度，存入文件名，存入文件内容。

对于文件内容 先存入 `hashMap`:先存入 `hashMap.size()`，再存入 `hashMap` 里的内容（这里的 `hashMap` 是 `hashMap<String,Byte>`）

存入文件字节的数目（方便之后解压读取的时候判断结束）

2.文件内容

读取文件，存入字符对应的 `huffmanCode`：

再次读取文件，每读到一个 Byte，找到它对应的 huffman 编码，存入一个 `stringBuilder` 里，`stringBuilder` 每满 8 位（即满一个 byte）就将它写入文件中，不足 8 位的再最后补 0 让它满足一个 byte，再写入文件中。

二．解压缩

读取压缩文件，

读取 `fileType`，如果是文件夹，读取包含文件的个数，读取文件名，在同级目录下创建文件夹，然后对文件夹里的文件进行操作，如果里面包含的还是文件夹，进行一个递归操作，继续读取文件夹。

如果是文件，读取文件名，读取 `hashMap`，读取文件的字节数，然后读取压缩文件里的内容，每次读取一个 Byte，将 byte 分成一个一个的 bit，定义一个 String，不断向 string 里加入 bit，当在 `hashMap` 里找到能找到原来的字符时，就将它写入文件里，然后让 `string=""`，直到写入的字节数等于读取的字节数停止（如果是文件夹，则开始下一个文件的读取）。

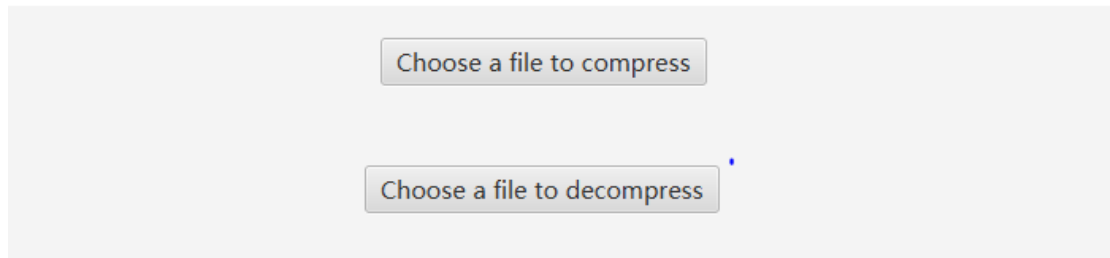
优化：主要对 IO 进行了优化，在实验中发现，调用库函数的 `read()`和 `wirte()`是一项非常耗费时间的操作，通过定义一个缓冲数组，将要写入的 byte 先存入缓冲数组里，再统一冲出。`read()`则使用了 `bufferedInputStream`，库函数自带的缓冲。

遇到的问题：存入字节数时，往往会超过 255，而读取的时候只能 readByte()，将字节数转换成 String 进行存储，读取的时候读取 String，再 ParseInt();

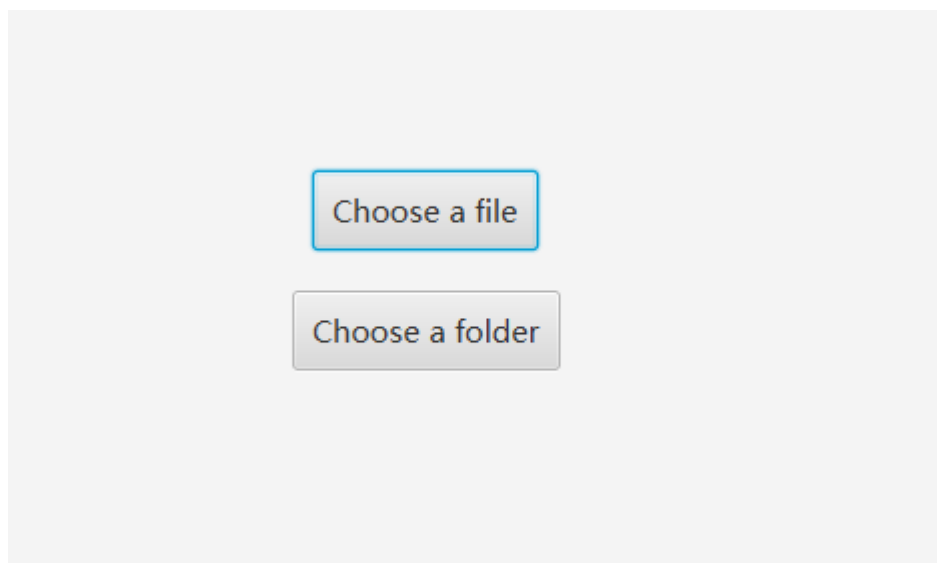
由于存入的时候存入了一个 hashMap.Size(),而 0~255 是 256 个字符，当遇到 size 刚好为 256 时 (发现 jpg, gif 等文件往往是 256 个字符都有)，写入 256, 由于 255=11111111, 所以读取 256 的时候出现了问题，于是又在 hashMapSize 前存了一个 flag 表示是不是 256 个字符。然而如果开始使用 byte 数组进行存储完全可以避免这个问题。

使用手册：

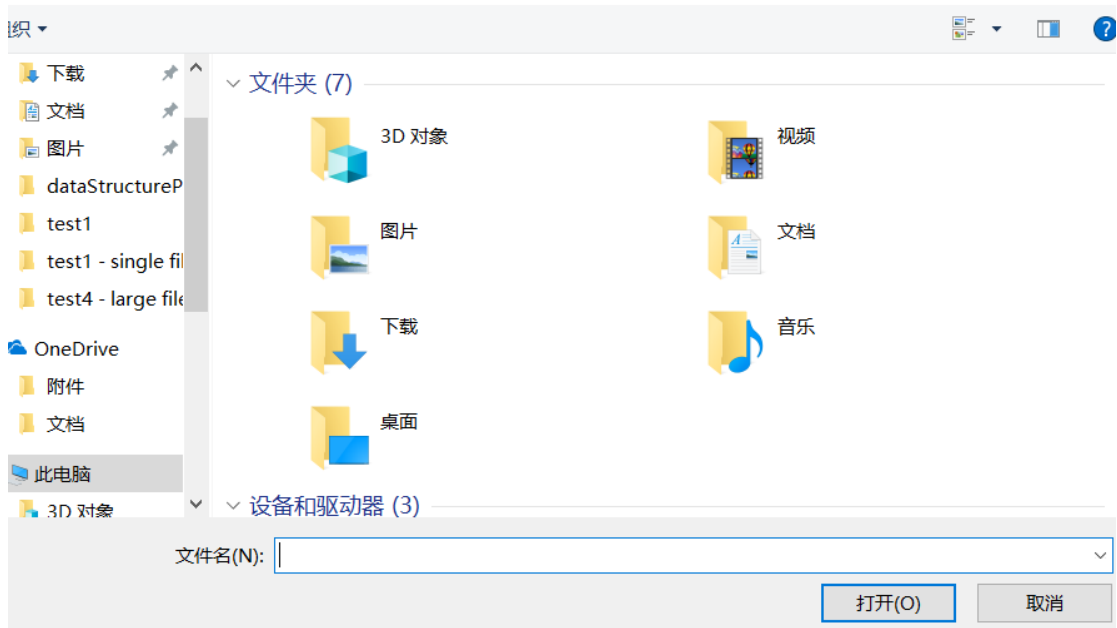
选择文件进行压缩或解压



Choose a file to compress:

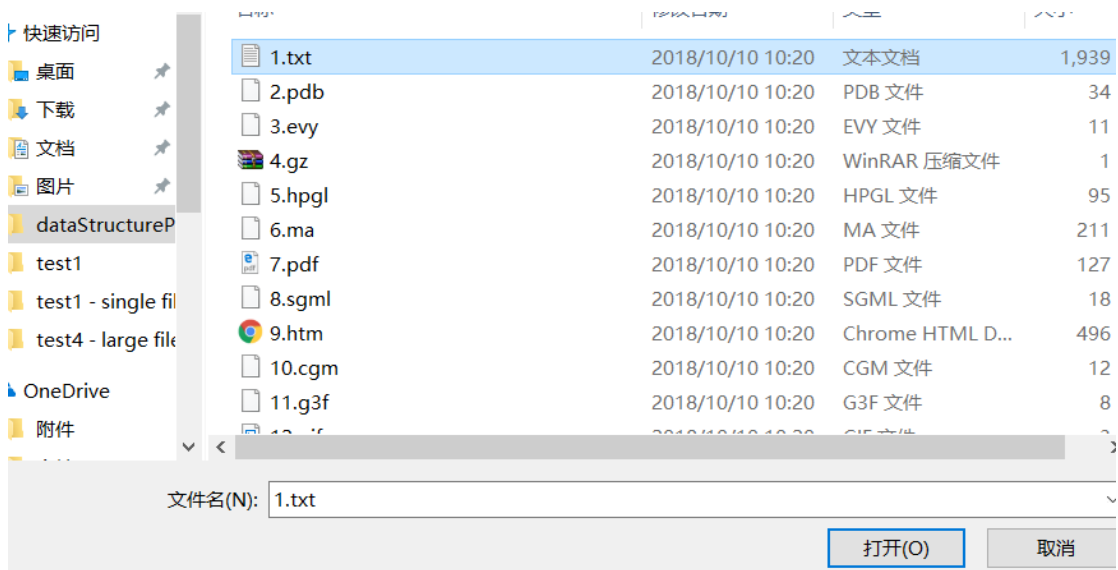


选择文件或文件夹：



结果展示：

压缩单个文件



压缩时间为：190ms

压缩文件

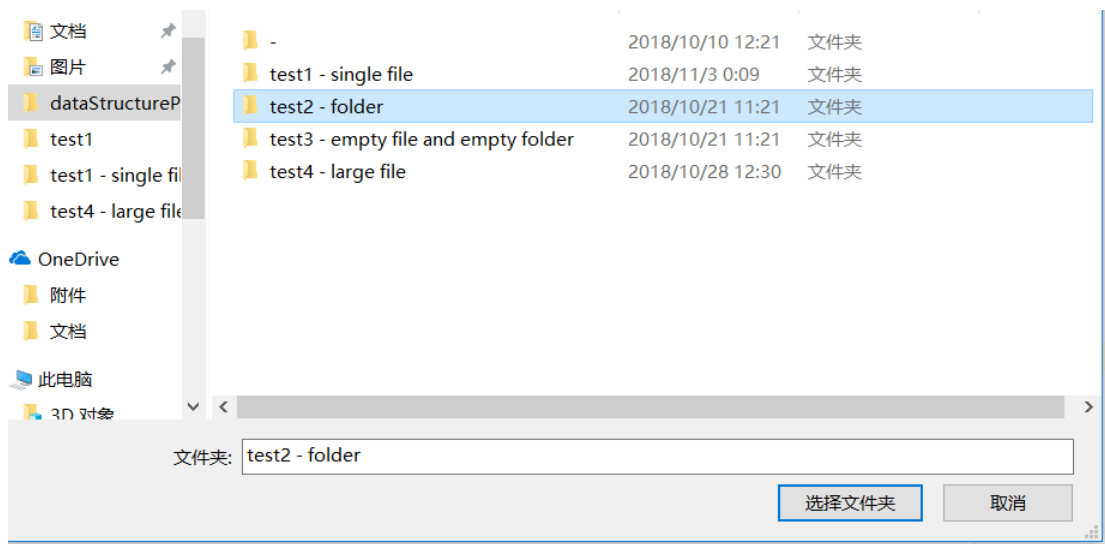
1.txt.yip 2018/11/3 14:26 VIP 文件 1,086 KB

压缩效率：56%

解压：

文件名	日期/时间	文件类型	大小
1.txt	2018/11/3 14:31	文本文档	1,939 KB
1.txt.yip	2018/11/3 14:30	VIP 文件	1,086 KB

压缩文件夹：



压缩后







test2 - folder.yip 2018/11/3 14:33 VIP 文件 9,220 KB








压缩效率: $9220 / (14.1 \times 1024) = 63.86\%$

解压



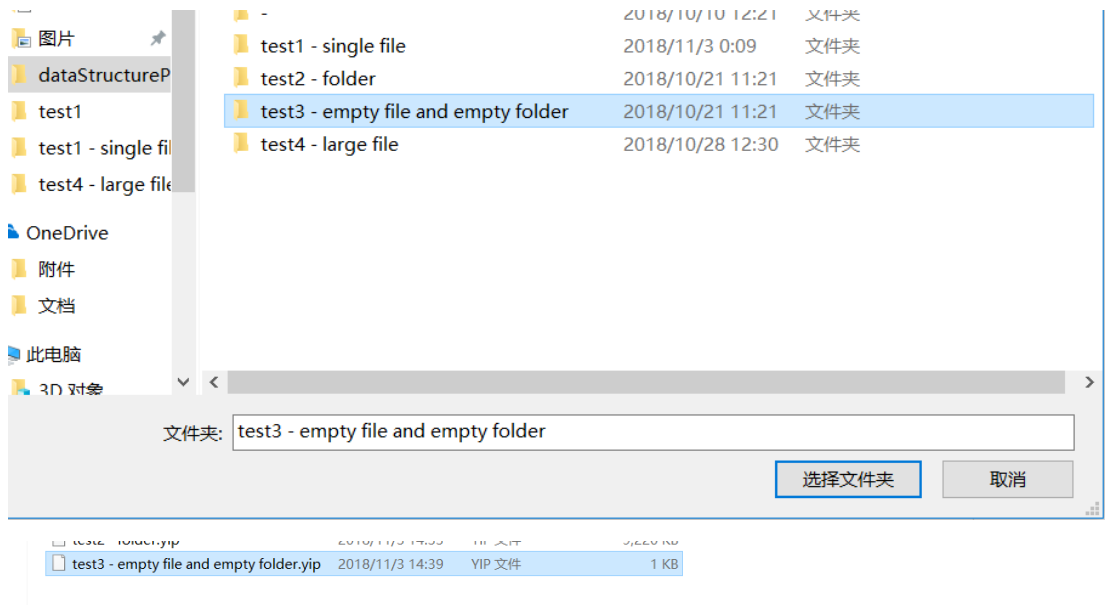
test2 - folder	2018/11/3 14:35	文件夹
1	2018/11/3 14:35	文件夹
2	2018/11/3 14:35	文件夹
3	2018/11/3 14:35	文件夹

5	2018/11/3 14:35	文件夹	
 7639141.htm	2018/11/3 14:35	Chrome HTML D...	75 KB
 7639143.htm	2018/11/3 14:35	Chrome HTML D...	58 KB
 7644487.htm	2018/11/3 14:35	Chrome HTML D...	69 KB
 7646300.htm	2018/11/3 14:35	Chrome HTML D...	115 KB
 7647024.htm	2018/11/3 14:35	Chrome HTML D...	53 KB
 7647499.htm	2018/11/3 14:35	Chrome HTML D...	47 KB

6	2018/11/3 14:35	文件夹	
 7791481.htm	2018/11/3 14:35	Chrome HTML D...	71 KB
 7796038.htm	2018/11/3 14:35	Chrome HTML D...	115 KB
 7796039.htm	2018/11/3 14:35	Chrome HTML D...	79 KB
 7796042.htm	2018/11/3 14:35	Chrome HTML D...	41 KB
 20030036425.htm	2018/11/3 14:35	Chrome HTML D...	221 KB
 20030167207.htm	2018/11/3 14:35	Chrome HTML D...	148 KB
 20030187787.htm	2018/11/3 14:35	Chrome HTML D...	74 KB

名称	修改日期	类型	大小
7750810.htm	2018/11/3 14:35	Chrome HTML D...	54 KB
7750812.htm	2018/11/3 14:35	Chrome HTML D...	141 KB
7755490.htm	2018/11/3 14:35	Chrome HTML D...	76 KB
7760094.htm	2018/11/3 14:35	Chrome HTML D...	160 KB
7760096.htm	2018/11/3 14:35	Chrome HTML D...	51 KB
7760097.htm	2018/11/3 14:35	Chrome HTML D...	170 KB
7762470.htm	2018/11/3 14:35	Chrome HTML D...	204 KB
7768379.htm	2018/11/3 14:35	Chrome HTML D...	342 KB
7786866.htm	2018/11/3 14:35	Chrome HTML D...	49 KB
7789314.htm	2018/11/3 14:35	Chrome HTML D...	54 KB
7789315.htm	2018/11/3 14:35	Chrome HTML D...	53 KB

压缩空文件夹和空文件



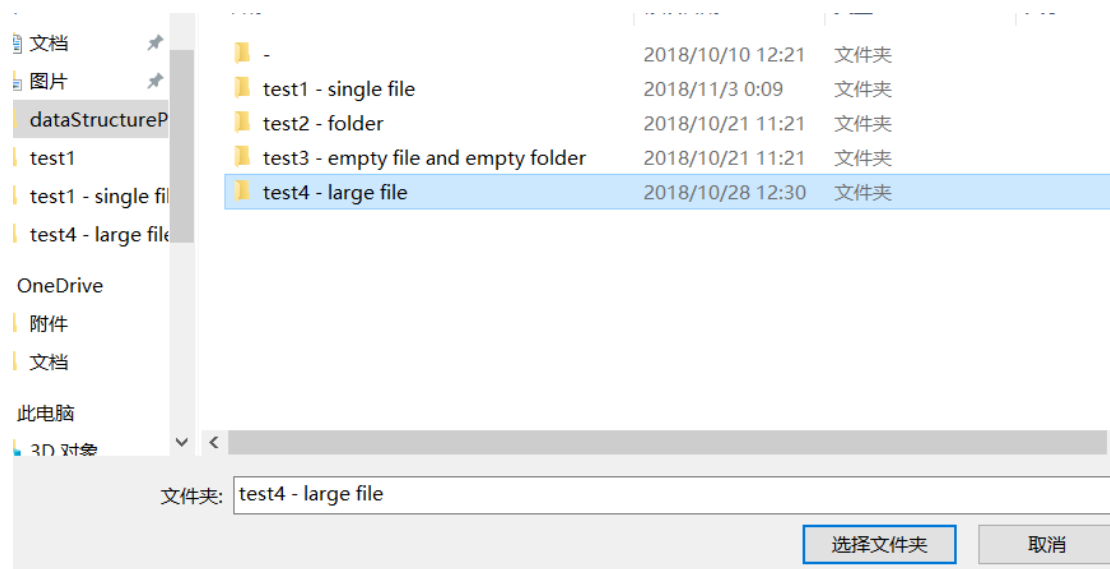
解压

test3 - empty file and empty folder	2018/11/3 14:40	文件夹	
empty_folder	2018/11/3 14:40	文件夹	
empty_file	2018/11/3 14:40	文件	0 KB

deaProjects > dataStructureProject > test3 - empty file and empty folder > empty_folder

名称	修改日期	类型	大小
该文件夹为空。			

压缩大文件：

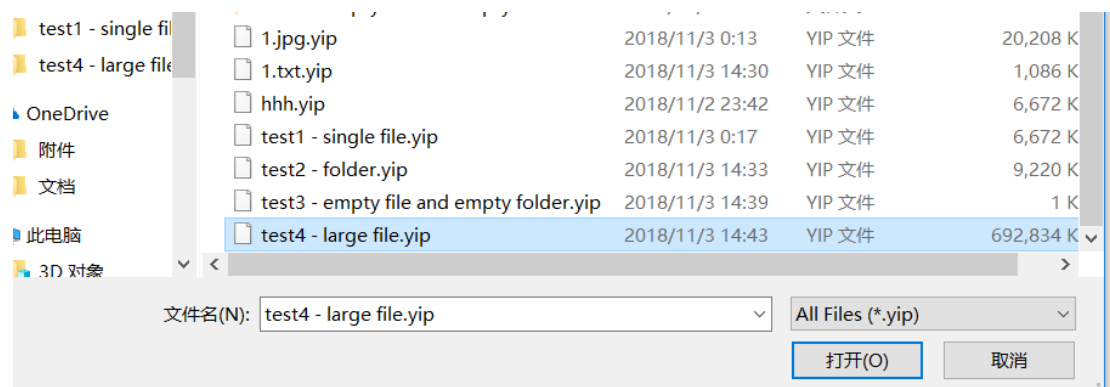


压缩时间大概: 1488+25335+29955 = 56778ms

test3 - empty file and empty folder.yip	2018/11/3 14:33	YIP 文件	1 KB
test4 - large file.yip	2018/11/3 14:43	YIP 文件	692,834 KB

压缩效率: $692834 / (1.02 \times 1024 \times 1024) = 64.78\%$

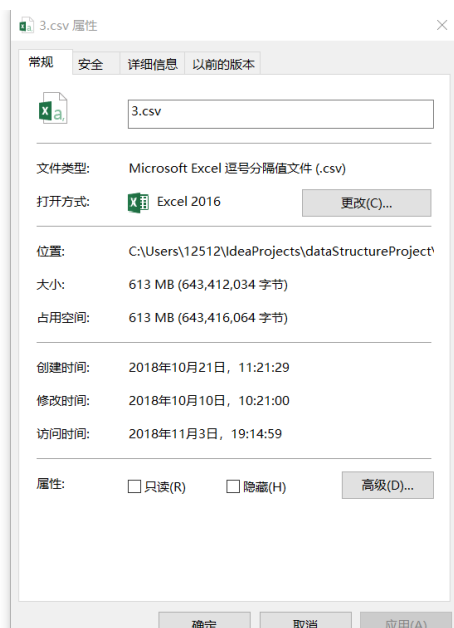
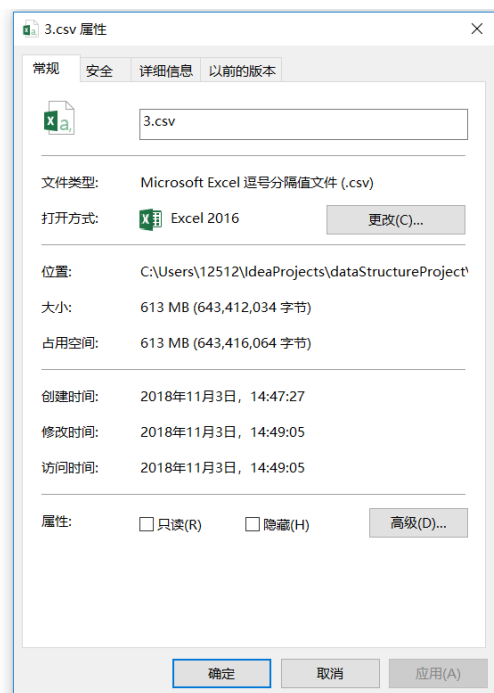
解压



解压时间: 165968ms = 2.766min



与原来对比, 字节数相等



与 winRAR 的对比：

测试文件		我的软件			
文件	文件大小	压缩后文件大小	压缩效率	压缩时间	解压时间
test1 - single file	8.74MB	6677kb	74.6%	986ms	1879ms
test2 - folder	14.1MB	9220kb	63.8%	1375ms	2617ms
test3 - empty file and empty folder	0kb	1kb	\	6ms	3ms
test4 - large file	1.02GB	692834kb	64.8%	55943ms	2.76min

测试文件		WinRAR			
文件	文件大小	压缩后文件大小	压缩效率	压缩时间	解压时间
test1 - single file	8.74MB	4408kb	49.3%	<1s	<1s
test2 - folder	14.1MB	3105kb	21.5%	1s~2s	1s~2s
test3 - empty file and empty folder	0kb	1kb	\	<1s	<1s
test4 - large file	1.02GB	143170kb	13.4%	57s	3s

winRAR 采用了更高级的压缩算法，压缩效率很高，尤其对表格，文档等类型的文件，压缩率远远超过我的，解压速度也很快，另外 winRAR 支持的操作也更多。