哈夫曼压缩开发文档

16307110325 朱小宁

基本思路：

首先需要建立一个哈夫曼树类，内部有个 Comparable 哈夫曼节点类。构建树的过程中需要找到当前 key 最小的节点，因此还需要一个二叉堆类。

压缩文件时，先按照一定格式写入压缩信息，包括编码表，再写入编码。

解压缩文件时，先按照设定的模式读取到压缩信息，构建编码表，再读取随后的哈夫曼编码，根据编码表还原文件。

实际实现：

```
core //核心代码
    dataStructure //数据结构
        CodeHuffTree //哈夫曼树
            //基本与课本相同
            //但是每个节点维护一个根据其位置变化的编码
        MinHeap //最小堆
    io //输入输出相关
        BitFileInputStream //按位读取的输入流
            //构造方法
            BitFileInputStream(BufferedInputStream inputStream)
            //内部调用的 BufferedInputStream
            private BufferedInputStream inputStream
            //当前读到的字节
            private byte thisByte
            //处于当前字节的哪个位置
            private int bitCount
            //返回数字 0 或 1，表示输入流的下一个 bit
            //每次读取一个 Byte，通过移位操作获取 bit
            int read()
        MyScanner //简化的无副作用的 Scanner
            //构造方法
            MyScanner(BufferedInputStream fileInputStream)
            //用来缓存字符
            private StringBuilder buf
            //所读取的输入流
            private BufferedInputStream fileInputStream
            //这个方法读取一串 byte，并转化为 UTF-8 编码的字符加入缓冲区
            //当碰到\n 符号时返回缓冲区，并将缓冲区清零
            //最终实现返回以\n 分隔的字符
            String nextLine()
    HuffmanCompress //含有压缩所需静态方法的类
        //压缩单个文件，需传入一个 256 长的数组作为编码表
```

//写入特定格式压缩信息和压缩编码
//调用上面所述的大部分类
private static void compressSingleFile(File inputFile, File outputFile,
int[][] codeTable, String path)
//压缩文件或文件夹，调用 compressSingleFile
private static void compressDir(File inputFile, File outputFile)
//压缩文件或文件夹，并重置静态变量，调用 compressDir
public static void compress(File inputFile, File outputFile)
//统计文件，获取文件编码数组
private static int[][] getFileCodeTable(File file)
HuffmanDecompress //含有解压所需静态方法的类
//读取压缩信息并根据其解压
public static void decompressFile(File zcsFile, File outputFile)
ui //javaFx 实现的 GUI
Main //运行时自动加载的主类

压缩文件格式示例：

```
##########################################################
$ZhuxiaoningCompressedFile
$TotalSize:110176
$FilePath:58PIC_tunyuntuwu_20090117459dd9532ff590a2.jpg
$FileSize:110176
$CodeTable:
1110111
10011101
11001101
……
11000010
11011001
0000001
$CodeHead
#编码区#
$CodeTail
$FilePath:hello.jpg
$FileSize:11302
$CodeTable:
111011111
100110101
……
110001010
11011001
0000001
$CodeHead
#编码区#
```

$CodeTail

##

碰到的问题：

1.在读取文件时同时用到了 Scanner 与 FileInputStream，发现 Scanner 的 next()方法具有副作用，它会在内部多次调用 FileInputStream 的 read()方法，导致文件流“指针”向后移动，使得调用 Scanner 的 next()方法后再调用 FileInputStream 的 read()方法会得到预料之外的结果。解决方法是：自己实现一个 MyScanner 类，模仿 Scanner 的部分行为，而且没有副作用。

2.建立哈夫曼树的过程中，首先考虑的是利用遍历叶子节点来获得其哈夫曼编码，但是发现这样不现实，因为这样需要遍历整个树，麻烦又慢。解决方法是：在节点中维护一个编码域，当一个节点被设置为其他节点的子节点的时候，其自身及其所有编码的前缀发生相应的增加(左 0 右 1)，这样，当哈夫曼树构建好之后，叶子节点的编码也计算好了。

3.程序运行异常缓慢，最后发现是 FileInputStream 的 available()方法太耗时间，减少使用后，速度提升 10 倍。

优化：

1.分析确定大部分时间开销都是 IO 比较慢造成的。为优化 IO 速度，应减少读写磁盘的次数，因此采用了带缓冲区的 IO 流进行读写，并根据文件大小动态确定缓冲区的大小。

2.减少字符串的使用，把不必要的 String 改为 char 或 int 数组。

3.减少字符串的连接，比如多次使用 print()而不是字符串连接后再 print()。

4.将 TreeMap 改为 HashMap 以改进查找时间。

5.将可以替换的 FileInputStream 的 available()方法替换。

测试&比较：

所有测试与比较中，压缩率指的是压缩文件大小相对原文件减小的百分比，压缩比指的是压缩文件大小与原文件大小之比。

测试一：单个文件

文件序号	压缩用时/ms	解压用时/ms	压缩率
1	250	641	43.98%
2	16	31	53.74%
3	16	31	-21.87%
4	0	15	-724.48%
5	0	32	51.97%
6	31	47	41.43%
7	15	70	18.78%
8	47	15	29.01%
9	94	156	30.52%
10	219	31	-11.31%
11	281	63	-19.49%
12	31	16	-78.48%

13	203	31	-11.22%
14	16	0	-156.30%
15	31	47	39.60%
16	16	15	15.04%
17	16	16	-11.84%
18	0	16	36.90%
19	31	78	34.64%
20	234	815	5.13%
21	31	47	-0.40%
22	93	282	58.97%
23	78	187	6.02%
24	16	16	33.22%
25	15	16	24.00%
26	15	16	7.97%
27	203	469	0.68%
28	47	16	-7.77%
29	15	0	-47.23%
30	0	16	######
31	0	31	5.69%
32	16	62	9.84%
33	47	109	12.03%
34	16	16	45.66%
35	0	31	4.95%

分析：很小的文件或已经压缩过的文件经过压缩会增大，因为要储存额外的压缩信息。

测试二：文件夹

文件序号	压缩用时/ms	解压用时/ms	压缩率
1	797	1663	37.80%
2	1563	2734	35.89%
3	328	907	34.79%

测试三：空文件和文件夹
（无错误发生）

文件名	压缩用时/ms	解压用时/ms	压缩后大小/byte
empty_file	32	10	362
empty_folder	4	2	40

测试四：

文件名	压缩用时/s	解压用时/s	压缩率
1.jpg	2.8	5.7	0.27%
2.csv	22.4	73.5	37.29%
3.csv	32.3	103.5	36.01%

比较一：小文件(参数默认)

测试文件： TestCase Test1-single file	本实现- zcs	WinRAR- rar	WinRAR- zip	好压- 7z
压缩比	74.5%	48.7%	54.8%	48.3%
压缩时间	<1s	<1s	<1s	5s

比较二：大文件(参数默认)

测试文件： TestCase Test4-large file	本实现- zcs	WinRAR- rar	WinRAR- zip	好压- 7z
压缩比	62.7%	13.5%	17.3%	13.1%
压缩时间	22s	54s	15s	370s

测试机配置

CPU: Intel(R) Core i5 7200U @ 2.50GHz
RAM: 8.00GB DDR4
Disk: LITEON T9 256