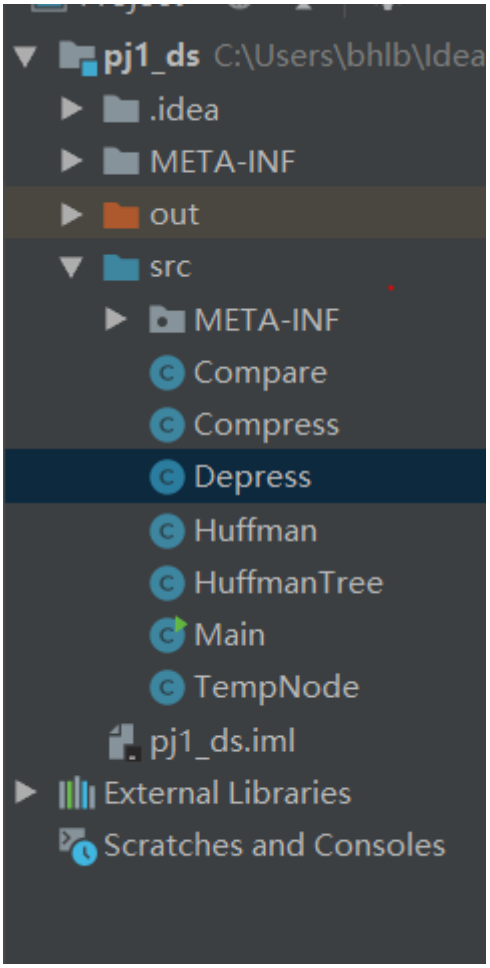
Development document

1．Design and implemention

Pj 主要分成了 7 个类。



Compare
+compare（HuffmanTree t1,HUffmanTree t2）：int

Compare 类通过重写 compare 方法来辅助实现优先队列。compare 方法实现 huffman 树之间的权重比较。

Compress
-PriorityQueue<HuffmanTree> +File inputFile,outputFile +BufferedOutputStream; +ObjectOutputStream;
+Compress(File inputFile,File outputFile) +compress():void +compress(File inputFile,File outputFile):void +close():void

Compress(File inputFile,File outputFile)重写构造方法。

compress（）方法重写 compress（File inputFile,File outputFile）方法，便于调用。
compress（File inputFile,File outputFile）是主方法，具体实现压缩功能。首先判断输入的文件是否存在，存在的话进而判断文件是文件夹还是单个文件。对于单个文件，用数字 4 标记，创建一个数组[TempNode]初始化每一种字符。然后通过 fieInputStream 读入文件，读入每种字符出现的次数，文件的长度。然后根据字符出现的次数排序，遍历每种字符出现的次数，如果字符出现的次数为 0，则表明字符没有出现过，记录前 i 个出现的字符种类。如果字符种类为 1，读入文件的名字，写入字符，字符的编码，出现的次数。如果字符种类大于 1，计算哈夫曼树所有结点的个数，每一种字符建立一个哈夫曼树，并记录文件的名字，写入字符，字符的编码，以及哈夫曼树在数组中出现的下标，并将数组的每一项记录到优先队列中。然后调用 createTree（）生成哈夫曼树，调用 hufCode()获取哈夫曼编码，在输出流中写入字符种类。初始化 code_buf 将生成的哈弗曼编码转化为二进制文件存入文件，最后关闭输入流输出流。对于文件夹，将文件夹里的所有文件存储在数组中，如果是空文件夹，用数字 3 标记，输出文件路径。如果不是空文件夹，对文件夹里面的每一个文件进行遍历，递归调用 compress。

Depress
-PriorityQueue<HuffmanTree> +File inputFile,outputFile +BufferedInputStream; +ObjectInputStream; +String
+Extract(File inputFile,File outputFile) +extract（）：void +check（String path）：void

Extract（File inputFile,File outputFile）重写构造方法，初始化一些全局变量。
extract（）是主方法，具体实现文件的解压，对于单个文件，还是分为字符种类是一种还是多种。得到压缩后的文件读入压缩后的文件得到字符出现的次数，重新构造 huffman 树，对于每个字符，从根向下知道叶子结点正向匹配对应的字符，按位与，最高位手机 1 向右走，否则向左，直到到达叶子结点，然后输出流里面写入文件相关信息。
函数 check（String path）方法主要是用来检查文件目录结构是否存在。

Huffman
+createTree （ HuffmanTree[] huf_tree, int char_kinds, int node_num, PriorityQueue<HuffmanTree> queue）：void +hufCode（HuffmanTree[] huf_tree, int char_kinds）：void

createTree（）方法用来构建 huffman 树。
hufCode（）方法通过遍历每一个字符种类获取 huffman 树的相关信息，左结点为 0，右节点为 1 以这样的形式保存每个节点的 huffman 编码。

HuffmanTree
+uch：byte +weight：int +code：String +index：int +parent：int +leftChild：int +rightChild：int

uch 是以八位为单位的字节，weight 是字节在文件中出现的次数，code 是字符相对应的 huffman 编码，index 记录的是结点在数组中的下标，parent，leftChild，rightchild 是结点。

Main
+main（String[] args）:void +compress():void +extract():void

Main 类是主类，用来运行压缩和解压。

main 方法使用递归根据输入内容调用 compress 和 extract 方法，实现压缩和解压。

compress 方法主要是获取压缩文件和解压文件的位置，调用 Compress 类的 compress 方法实现压缩

extract 方法主要是获取压缩文件和解压文件的位置，调用 Extract 类的 extract 方法实现对压缩文件的解压。

TempNode
+uch：byte +weight：int
+compareTo（TempNode tempNode）：int

这个类主要是实现节点之间的比较。

2.performance of test cases

文件	压缩率(%)	压缩时间/ms	解压时间/ms
1.txt	56.08	391	187
2.pdb	45.24	63	47
3.evy	100	47	16
4.gz	100	47	15
5.hpgl	46.85	79	62
6.ma	58.57	172	62
7.pdf	81.74	125	62
8.sgml	67.97	62	16
9.htm	69.69	203	109
10.cgm	100	47	16
11.g3f	100	32	15
12.gif	100	31	15
13.jpg	100	63	32
14.png	100	47	16
15.ps	60.09	109	63
16.svg	74.52	31	15
17.tif	100	47	16
18.xbm	100	32	31
19.msh	66.35	172	78
20.mov	95.62	438	312
21.mpeg	100	93	63
22.igs	41.31	297	110

23.v5d	94.20	218	110
24.wrl	63.93	47	47
25.aiff	69.26	63	16
26.au	85.68	141	15
27.mp3	100	328	235
28.ra	100	63	15
29.wav	100	31	16
30.ram	100	16	16
31.aiff	91.74	78	31
32.aiff	89.65	156	78
33.aifc	88.12	203	110
34.tsv	51.02	31	15
35.avi	91.29	63	32
1（文件夹）	60	906	813
2（文件夹）	53.98	2828	1547
3（文件夹）	63.2	875	829
mpty-file	/	15	0
1.jpg	100	3203	2813
2.csv	62.94	66257	33800
3.csv	64.27	92087	49438

3.problems

1. 文件压缩时间太长，在 FileInputStream，FleOutputStream 的基础上使用 BufferedInputStream，BufferedOutputStream 建立缓冲区，将数据储存在这里，避免每次都在硬盘上读取，提高文件读取速率。

2.无法实现文件夹的压缩，通过区分文件是单个文件还是文件夹，在文件和文件夹后面用不同的数字标记。对文件夹中的文件进行遍历，递归调用文件的压缩函数。

4.techniques

1.使用 BufferedInputStream，BufferedOutputStream 提高文件的读取速率。

5.comparison

文件	压缩率（winrar vs huffman）	压缩时间(winrar vs huffman)
2.csv	17%：62.94%	42s　:66.25s

压缩率与 winrar 有着很大的差距，压缩时间也有着一定的差距。

Winrar 解压时间只需要 1s，差距很大很大。