游程编码

1900012476 夏罗生

游程编码

游程编码（RLC，Run Length Coding），是一种无损编码。

其优点在于将重复性高的数据压缩成小单位；由于读取数据，压缩数据，写入数据可以同时进行，所以不需要占用过多内存，不受文件大小的限制可以无限压缩下去。

其缺点在于不能随机访问，如果数据的重复频率不高，则压缩结果可能比原数据还大。

数据结构

相比于使用抽象基类，我认为使用类模板实现更加简单。

当文件读入到内存中时，不论是从原始文件读入，还是从压缩文件读入，在内存中的存储形式都是相同的。声明一个结构模板，一个结构存储一个元素和其重复的次数。

```
template <class T>
struct RepeatSymbol {
  T value;
  int count;
  bool is_head;  //头元素只是一个占位元素，用来初始化第一个元素
  RepeatSymbol(T v, int c, bool h) : value(v), count(c), is_head(h) {}
};
```

声明一个模板类，使用一个结构数组储存所有数据。

```
template <class T>
class RunLengthCoder {
 private:
  //文件内容的行数和列数，用于解码时还原文本文件
  int rows, columns;
  //原始文件和编码文件的大小，以字节为单位
  int decode_file_size, encode_file_size;
  //编码数组
  vector<RepeatSymbol<T>> symbols;

 public:
  //构造函数
  RunLengthCoder();
  //从原始文件中读取数据，并存储到内存中
  void ReadDecodedFile(string filename);
  //编码后写入文件
  void OutputEncodedFile(string filename);
  //从编码文件中读取数据，并存储到内存中
  void ReadEncodedFile(string filename);
  //解码后写入文件
  void OutputDecodedFile(string filename);
  //判断两个元素是否相等
  bool IsEqual(T cur, T next);
  //写入文件后，输出压缩信息
```

```
25      void PrintCompressionInfo();
26  };
```

值得注意的有两点。

首先对于浮点数，不能使用简单的 == 运算符判断相等，所以添加了一个模板函数 `bool IsEqual(T cur, T next)`，用于判断数据类型是否相等。

其次，对于对于二进制文件的读取，使用 `unsigned char` 表示，由于二进制文件的读取机制和文本文件不同，所以需要进行显式具体化，进行额外定义。

编码策略

其编码策略和输出格式要分为文本文件和二进制文件两种情况讨论。

文件在计算机上都是以二进制存储，所以文本文件与二进制文件的区别并不是物理上的，而是逻辑上的。这两着只是在编码层次上有差异。

简单来说，文本文件是基于字符编码的文件，常见的编码有 ASCII 编码，UNICODE 编码等等。二进制文件是基于值编码的文件，你可以根据具体应用，指定某个值是什么意思（这样一个过程，可以看作是自定义编码）。

文本文件

在输入文件中，第一行会给出数据的行列数，之后为具体数据。输出文件是文本文件，所以可以直接使用一个整型表示重复数。

由于作业要求当新字符出现时需要换一行，已经造成了很多的空间浪费，所以采用默认的编码策略。

同时为了做到无损压缩，需要在编码时将行列数也写到文件中，否则无法还原原始文件。

二进制文件

对于二进制文件，不能简单用一个整型表示重复数。因为一个整型在二进制文件中是以位存储的，在编码时一定要确定一个整型占据多少位，才能在解码时准确地得到重复数的大小。

假如说使用 INT32 类型存储，则每一个重复数都要占据 32 位，非常浪费内存。假如说使用 Char 类型存储，则当重复数大于 255 时无法存储。

采取的编码策略为 4 位表示法，使用 4 位来存储整数，当重复次数超过 15 时，重新开始一次计数。

压缩率

使用压缩文件的大小除以原始文件的大小的结果作为压缩率。