Huffman 编码

詹江岳

Date: 2017-12-2

Chapter 1: Introduction

问题描述：对目标文件进行 Huffman 编码压缩，并且生成的压缩文件能够解压得到和目标文件一样的内容。

算法背景：1951 年，哈夫曼和他在 MIT 信息论的同学需要选择是完成学期报告还是期末考试。导师 Robert M. Fano 给他们的学期报告的题目是，寻找最有效的二进制编码。由于无法证明哪个已有编码是最有效的，哈夫曼放弃对已有编码的研究，转向新的探索，最终发现了基于有序频率二叉树编码的想法，并很快证明了这个方法是最有效的。由于这个算法，学生终于青出于蓝，超过了他那曾经和信息论创立者香农共同研究过类似编码的导师。哈夫曼使用自底向上的方法构建二叉树，避免了次优算法 Shannon-Fano 编码的最大弊端——自顶向下构建树。(百度百科)

哈夫曼单个字符的编码长度和该字符在特定文件中的信息熵基本相同（是一个向上取整的关系），因此哈夫曼编码是十分有效的。

Chapter 2: Algorithm Specification

- 主要数据结构设计说明

1. heap

逻辑上是一个完全二叉树，采用一维数组实现。

树的每个内部结点的优先级都比它的左右两个儿子要高。

主要操作	方法说明
bool insert(const T& element)	将 element 插入堆
const T& top() const	取出堆顶的元素
void pop()	删除堆顶的元素
T pull()	获取堆顶元素并将其删除
int size() const	获取堆内元素数量

2. HuffmanTree

逻辑上是一个满二叉树，采用二叉树的链式实现。内部节点有指向左右儿子的指针，叶结点存储单个字符。用于构建编码二叉树的结点还带有权重，而只用于解码的结点不带权重。

编码二叉树的带权路径长度最小。带权路径长度：从根结点到该结点之间的路径长度与该结点的权的乘积。即所有叶结点的带权路径长度之和最小。

主要操作	方法说明
node* getRoot() const	获取哈夫曼树
const code* getTable() const	获取哈夫曼编码表
unsigned __int64 getCharLength() const	获取当前文件在哈夫曼编码下的大小
void getTreeInfo(treeInfo& infoB) const	获取哈夫曼树信息，包括它的前序序列化字符串

- 系统设计思想：

  见《附件 1_类图.jpg》。

- 程序流程图

见《附件 2_流程图. jpg》

Chapter 3: Testing Results

模块	测试文件	测试内容	预期结果	实际结果	预计原因	实际原因	现状
英文字符文档	1.txt	压缩到1.hf	生成1.hf的压缩文件	生成1.hf的压缩文件			pass
	1.hf	解压到1-1.txt	生成1-1.txt的文档，内容与1.txt一致	崩溃	文件头书写或解析错误	nodeRecordLength的头记录解析错误	pass
				解压文件最后一个字符不对	lastSigBitNum的写入有问题	lastSigBitNum写入前没有对齐偏移量	pass
单字符文档	2.txt	压缩到2.hf	生成2.hf的压缩文件	生成2.hf的压缩文件			pass
	2.hf	解压到2-1.txt	生成2-1.txt的文档，内容与2.txt一致	生成2-1.txt的文档，内容与2.txt一致			pass
含中文字符的文档	3.txt	压缩到3.hf	生成3.hf的压缩文件	生成3.hf的压缩文件			pass
	3.hf	解压到3-1.txt	生成3-1.txt的文档，内容与3.txt一致	生成3-1.txt的文档，内容与3.txt一致			pass
大文件	idealU-2017.1.3.exe	压缩到4.hf	生成4.hf的压缩文件	生成4.hf的压缩文件			pass
	4.hf	解压到4.exe	生成4.exe，MD5值与idealU-2017.1.3.exe一样	生成4.exe，MD5值与idealU-2017.1.3.exe一样			pass

Chapter 4: Analysis and Comments

- 算法分析：

  时间复杂度：生成编码表的过程需要遍历一次文件，遍历一次哈夫曼树，然后在进行哈夫曼编码时再遍历一次文件。因此，若将 n 设为输入文件的大小，整个压缩过程的时间复杂度是Θ(n)。

  解压过程先要读取文件头，哈夫曼树的序列化记录，重构二叉树，这个过程可视为常数，解压时要对目标文件写入 n 个字符，即压缩前文件是 n 个字节大小。因此，解压过程的时间复杂度是Θ(n)。

  空间复杂度：压缩时算法最主要的内存开销是保存编码表，不论原文件是多大，都会生成长度为 256 的编码表，主要的变化是编码长度引起的动态内存分配，这个总体上不会超过某一常数(32895?)。因此，压缩的空间复杂度是Θ(1)。

  解压的空间开销主要是哈夫曼树，而哈夫曼树的结点数不会超过 511。因此，解压的空间复杂度也是Θ(1)。

- 算法特色：

  生成编码表即重构哈夫曼树的过程扁平化，即利用栈实现，而没有递归。

- 不足：

  程序的容错率低，基本没有错误处理。另外，设计上应该是可以支持多线程压缩的，后续可以尝试实现。

Declaration

I hereby declare that all the work done in this project titled "Linear List" is of my independent effort as an individual.

Duty Assignments:

Programmer: 詹江岳

Tester: 詹江岳

Report Writer: 詹江岳