

基于 Huffman 编码的压缩算法的实现

李钰 19335112 liyu256@mail2.sysu.edu.cn

摘要

概述：本实验为运用 Huffman 编码规则对文件进行编码，以实现压缩文件和解压文件。

解决方法

压缩：①统计字符出现频率

②构建 Huffman 树

③创建针对该待压缩文件的 Huffman 编码

④字符转换并存入输出文件中

解压：①读取原文件的字符出现频率

②构建 Huffman 树

③构建原文件 Huffman 编码规则

④字符转换存入解压文件中

结论：该程序只适用于压缩 ASCII 码中所包含的字符，且只能顺利压缩文本文件，其他格式的文件没有实现成功。对于小文件压缩和解压速度较快，但稍大一点的文件用时很久。

一. 引言

解决的问题：利用 Huffman 编码进行文件压缩和解压

解决方法：构造最优二叉树，创建 Huffman 编码规则，将字符转换为 01 串进行存储

通过该实验可深入了解 Huffman 编码原理，体会其压缩原理和空间节省的优点；熟悉二叉树结构体，掌握其构造方法；熟悉和掌握文件的基本读写操作。

二. 解决方法

1. 输入形式：

首先通过输入数字（1、2、3）来做出操作选择；

接着输入相应的文件名（注意：这里要加文件扩展名）

2. 输出形式：输入文件名后，可看出相应的操作反馈：如“Compress successfully!”等字样，并提示用户进行下一步操作。

3. 使用的数据结构：结构体 二叉树 数组 向量

①结构体

本程序创建的结构体中数据成员有该节点的字符，对应字符的出现频率，以及左右两个子节点；成员函数为该结构体的构造函数，对新建节点初始化。

创建该结构体的目的是为构建 Huffman 树时新建节点，存储字符及其权重。

```
typedef struct node{
    char val;//记录字符
    int rate;//出现频率
    node *left, *right;

    node(){ //construction,-1表示无字符
        val = -1;
        rate = 0;
        left = NULL;
        right = NULL;
    }
}Node;
```

②二叉树

构建 Huffman 树时使用二叉树的数据结构

③数组

本程序中共使用两个数组

`int frequency[256]`用于统计字符出现频率，数组大小为 256，即所有 ASCII 码的个数，这样可以用数组下标来对应字符大小，方便简洁。

`string Huffman_code[256]`用于存储下标值对应字符的 Huffman 编码

④向量

构建向量，其中元素是节点。建树时容纳所有节点，以便查找未加入二叉树的节点中的权重最小的那个，以及后续节点的增添，方便了树的构建。

4. 算法：

压缩函数

Step1 记录文件扩展名及其大小，并写入输出文件中，以便后续解压时获取原文件的文本类型

Step2 统计原文件各自符的出现频率，写入输出文件中。具体实现为循环遍历整个文件的所有字符，每遍历到一个字符，则对统计频率数组中下标与字符相等的那个元素值加一。

Step3 利用频率统计的数组，构建哈夫曼树。

具体过程

①先将数组值（及统计的出现频率）不为 0 的对应字符节点存到 `vector` 容器中

②若 `vector` 的 `size` 值为 0，则直接返回空指针

③当 `vector` 的 `size` 大于 1 时进行循环：查找、记录并删除 `vector` 中节点 `rate` 值最小的两个节点，以这两个节点为子结点，新建节点，该新建节点的 `rate` 值为两个子节点的 `rate` 值之和，并将该节点加入到 `vector` 容器中。

④当 `vector` 的 `size` 等于 1 时，该 `vector` 中的唯一节点即为我们构建的 Huffman 树的根

Step4 通过 Huffman 树得到 Huffman 编码

具体过程

因为要编码的字符都在叶子节点的位置，所以采用递归函数的思想，一直向左遍历子树，每遍历一个子树，则 `code` 字符串+“0”，直到其没有子树（即已经到达叶子节点），得到该叶子节点字符的 Huffman 编码；然后 `code` 串擦除末尾字符，回溯到父亲节点，再向右遍历子树，这时每遍历一个子树 `code`+“1”，直到到达叶子节点

到达叶子节点时，把对应叶子结点的 Huffman 码存入到 `Huffman_code` 数组对应的位置

Step5 对待压缩文件进行 Huffman 码编码，生成 01 字符串，将该串的大小写入输出文件，然后将该串 8 位 8 位的转换为 `char` 类型字符串并写入到输出文件中

压缩完成

解压文件

Step1 读出原文件扩展名并更新输出文件名字

Step2 读取频率表

Step3 通过频率表构建 Huffman 树

Step4 通过 Huffman 树构建适用于原文件的 Huffman 编码

Step5 读取原文件进行 Huffman 编码后的大小以及转换后的字符串

Step6 翻译字符串，写入到输出文件中

解压完成

5. 函数

```
void menu();//界面菜单
void frequency_count(string, int*);//频率统计
vector<Node*>::iterator find_min(vector<Node*>&);//寻找权重最小值
Node* construction_tree(int*);//构建Huffman树
void construction_code(Node*, string, string*);//构建Huffman码
char transformationTochar(string, int);//将01串转换为char
void destruction(Node*);//删除树
void compression(string, string);//压缩
Node* transformation2(Node*, Node*, char, int&, int, char*, int&);//转换翻译
void decompression(string, string&);//解压 这里注意解压后的文件扩展名未知, 要进行添加修改, 所以按引用传递参数
```

三. 程序使用和测试说明

程序运行后显示主界面

```
Please enter your choice
1.compress file
2.decompress file
3.exit
```

输入相应操作, 如: 用户想要压缩, 则输入 1

这时系统提示继续输入要压缩的文件名, 注意这里要加文件类型的后缀

```
Please enter your choice
1.compress file
2.decompress file
3.exit
1
Please enter the file name with file extension. (eg:lalala.txt)
_
```

输入形式正确且存在的文件名之后, 程序进行压缩操作, 完成压缩后给予反馈, 出现“Compress successfully!”字样, 并提示接下来的操作

```
Please enter your choice
1.compress file
2.decompress file
3.exit
1
Please enter the file name with file extension. (eg:lalala.txt)
test1.txt
Compress successfully!
-----
Please enter your choice
1.compress file
2.decompress file
3.exit
```

接着测试解压, 输入 2, 以及待解压的文件名, 提示“Decompress successfully!”, 解压成功

```

-----
Please enter your choice
1.compress file
2.decompress file
3.exit
2
Please enter the file name with extension. (eg:lalala.bin)
test1.bin
Decompress successfully!
-----
Please enter your choice
1.compress file
2.decompress file
3.exit

```

最后输入 3，退出系统

```

-----
Please enter your choice
1.compress file
2.decompress file
3.exit
3
Thanks for your using!

```

对于容错性的讨论，假若输入的文件不存在，系统提示：打开文件时出错，程序结束







```

Please enter your choice
1.compress file
2.decompress file
3.exit
1
Please enter the file name with file extension. (eg:lalala.txt)
test1.txt
Open file error!
Program end

Process returned 0 (0x0)   execution time : 123.221 s
Press any key to continue.

```

最终压缩效果

 test1.bin	2020/10/25 9:32	BIN 文件	43 KB
 test1	2020/10/25 9:32	文本文档	79 KB
 test2.bin	2020/10/25 10:44	BIN 文件	210 KB
 test2	2020/10/25 10:45	文本文档	373 KB
 test3.bin	2020/10/25 11:17	BIN 文件	348 KB
 test3	2020/10/25 11:25	文本文档	649 KB

四. 总结和讨论




特色：所有数组大小为 256，因为 ASCII 码表内的字符共 256 个，下标值即可对应相应 ASCII 码表中的值；压缩时将原文件转换的 Huffman 码每 8 位转换为 char 类型存到输出文件中，降低内存空间损耗

问题：能力有限，测试时发现这个程序只对文本文件能起到压缩作用，其他格式的文件都会压缩不成功，并且当文本文件过大是压缩时长会明显增加，测试 test2 压缩时花费了大

概 3~4 分钟的时间，test3 则更久。

原程序上的一点小改进

因为一开始的程序解压文件的文件名和原文件名相同，覆盖了原文件里的内容，所以无法比对解压之后的文件是否满足“无损解压”，于是修改了解压时的输出文件名，在原文件名后加了_ed,测试 test1，得到下面结果

 test1.bin	2020/10/25 16:33	BIN 文件	43 KB
 test1	2020/10/25 15:44	文本文档	79 KB
 test1_ed	2020/10/25 16:33	文本文档	79 KB

比对后发现和原文件相同，未被修改

通过该程序收获最大的应该是读写文件方面，例如 open 函数的第一个参数必须是 char* 类型，所以用之前要通过 c_str() 函数将 string 类转换为 char*，而 c_str() 的返回是 const char*；以及二进制文件的读写操作要用到 read 和 write 函数，借此机会从学习了该函数的用法。

除此之外更进一步熟悉了 Huffman 编码，以及如何构建最优二叉树，如何利用递归函数确定字符对应的 Huffman 码。

之前学到了 auto 的用法，可以使我们的程序更简洁，但在 code blocks 和 DEV 上用会报错而 VS code 上不会，之后要研究一下原因。

该实验还第一次使用了 exit(0)，包含在<algorithm>头文件中。

五. 参考文献

<http://cplusplus.com>

<https://blog.csdn.net>

<https://blog.csdn.net/yshshhgq/article/details/83854606>

《离散数学基础》乔海燕 周晓聪