# GZIP之huffman编码的文件压缩

## [本节目标]

- **1. 引言**
- **2. huffman树**
- **3. huffman树构建**
- **4. 获取huffman编码**
- **5. 压缩**
- **6. 压缩文件格式**
- **7. 解压缩**
- **8. 测试**
- **9. 关于huffman的一些思考**



原理：给每个字节找一个更短的编码（二进制比特位）

## 1. 引言

> **GZIP的第一步是使用LZ77快速的对文件中的重复的短语进行压缩**，压缩完成之后，就形成了一个包含原字符、距离、长度的压缩文件，该文件中重复的语句基本已经被全部消除掉，但是各个字节上还有一定程度的重复，因此**GZIP的第二步就是从字节方面来进行压缩的。**
>
> **一个字节占用8个比特位，如果能够给一个字节找到更短的编码，即少于8个比特位，就可以起到压缩的目的**，编码一般分为：**静态等长编码和动态不等长编码。**
>
> 比如：ABBBBCCCCCDDDDDDD字符串
>
> 1. **静态等长编码**：
>
>    **文件中共有4个不同种类的字符，因为每个字符可以用两个二进制的比特位表示**

| 字符 | 编码 |
| --- | --- |
| A | 00 |
| B | 01 |
| C | 10 |
| D | 11 |

用等长编码对上述源数据进行压缩：01101110 11110111 11100011 10011110，压缩完成后的结果只占4个字节，压缩率还是比较高的。

该种压缩方式一般要求文件中字符种类比较少，但是一般情况下文件中字节的种类是比较多的。

2. **动态不等长编码**

根据文件中字节的分布情况获取每个字节的编码。

| 字符 | 编码 |
| --- | --- |
| A | 100 |
| B | 101 |
| C | 11 |
| D | 0 |

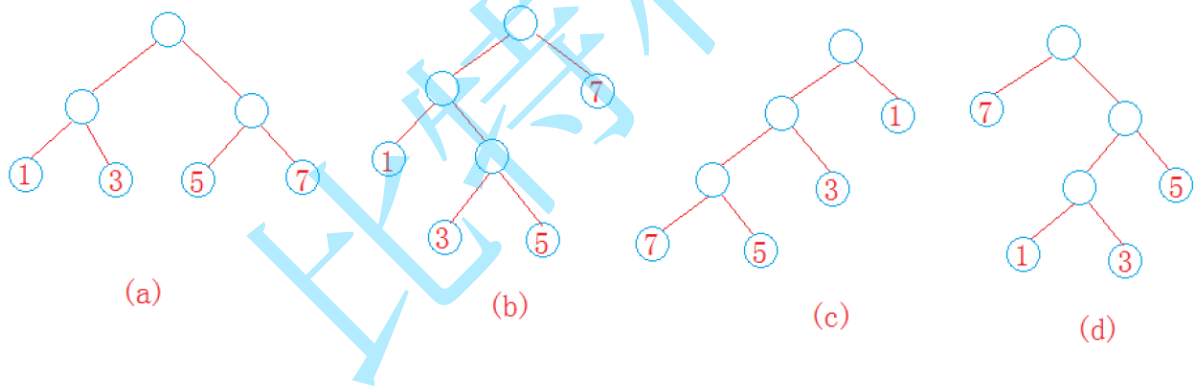
使用不等长编码对源数据进行压缩：10111011 00101001 11000111 01011

压缩完成后最后一个字节没有用完，还剩余3个比特位，对于该文件中内容，动态不等长编码方式比等长编码方式的压缩率能好点。

上述动态不等长编码有一种方式可以简单获取到，huffman树。

## 2. huffman树

**从二叉树的根结点到二叉树中所有叶结点的路径长度与相应权值的乘积之和为该二叉树的带权路径长度WPL。**




具有相同叶结点和不同带权路径长度的二叉树

上述四棵树的带权路径长度分别为：

- **WPLa = 1 * 2 + 3 * 2 + 5 * 2 + 7 * 2 = 32**
- **WPLb = 1 * 2 + 3 * 3 + 5 * 3 + 7 * 1 = 33**
- **WPLc = 7 * 3 + 5 * 3 + 3 * 2 + 1 * 1 = 43**
- **WPLd = 1 * 3 + 3 * 3 + 5 * 2 + 7 * 1 = 29**

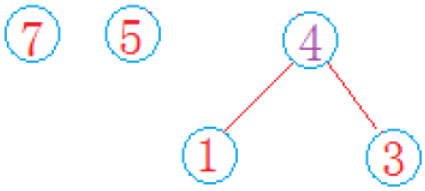
**将带权路径最小的二叉树称为Huffman树。**

## 3. huffman树构建

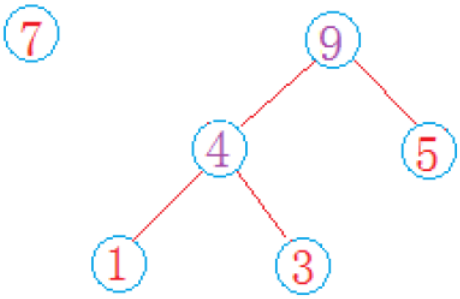
1. **由给定的n个权值{ w1, w2, w3, ... , wn}构造n棵只有根节点的二叉树森林**F={T1, T2 , T3, ... ,Tn},每棵二叉树Ti只有一个带权值wi的根节点，左右孩子均为空。
2. 重复以下步骤，直到F中只剩下一棵树为止

- 在F中选取两棵根节点权值最小的二叉树，作为左右子树构造一棵新的二叉树,新二叉树根节点的权值为其左右子树根节点的权值之和
- 在F中删除这两棵二叉树
- 把新的二叉树加入到F中

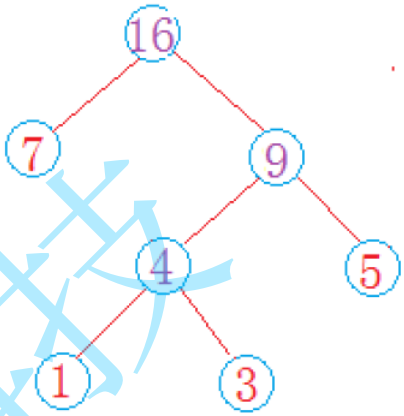

第一步




第二步




第三步



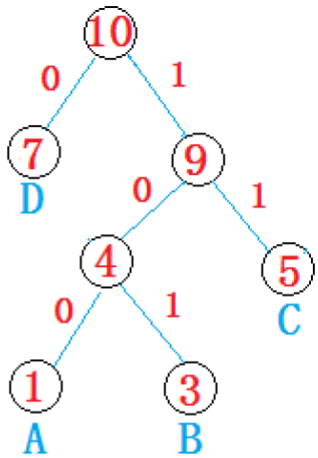

第四步



## 4. 获取huffman编码

BCDCDDBDDCADCBDC

A->1 B->3 C->5 D->7




1. 以字符串中每个字符出现的总次数为权值构建huffman树
2. 另huffman树中左分支用0代替，右分支用1代替
3. 所有权值节点都在叶子位置，遍历每条到叶子节点的路径

获取字符的编码

A:100

B:101

C:11

D:0

堆：元素存储在一个一维数组中（完全二叉树）
且每个节点都比其孩子节点大（大堆）或者小（小堆）

**问题：如果不等长编码中出现一个编码是另一个编码的前缀怎么办？该情况是否会出现?**

## 5. 压缩

1. 打开被压缩文件，获取文件中每个字符串出现的总次数。
2. 以每个字符出现的总次数为权值构建huffman树。

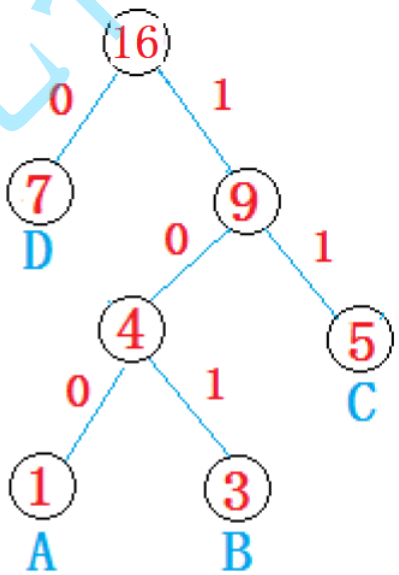
3. 通过huffman树获取每个字符的huffman编码。
4. 读取源文件，对源文件中的每个字符使用获取的huffman编码进行改写，将改写结果写到压缩文件中，直到文件结束。

## 6. 压缩文件的格式

假设按照上述操作已经对源文件压缩完毕，怎么解压缩？解压缩之后文件的后缀怎么与源文件保持一致？因此：**压缩文件中除了保存压缩数据外，还必须保存解压缩所需的信息。**压缩文件格式：


## 7. 解压缩

1. 从压缩文件中获取源文件的后缀
2. 从压缩文件中获取字符次数的总行数
3. 获取每个字符出现的次数
4. 重建huffman树




5. 解压压缩数据
    - a. 从压缩文件中读取一个字节的获取压缩数据ch
    - b. 从根节点开始，按照ch的8个比特位信息从高到低遍历huffman树：

- 该比特位是0，取当前节点的左孩子，否则取右孩子，直到遍历到叶子节点位置，该字符就被解析成功
- 将解压出的字符写入文件
- 如果在遍历huffman过程中，8个比特位已经比较完毕还没有到达叶子节点，从a开始执行

c. 重复以上过程，直到所有的数据解析完毕。

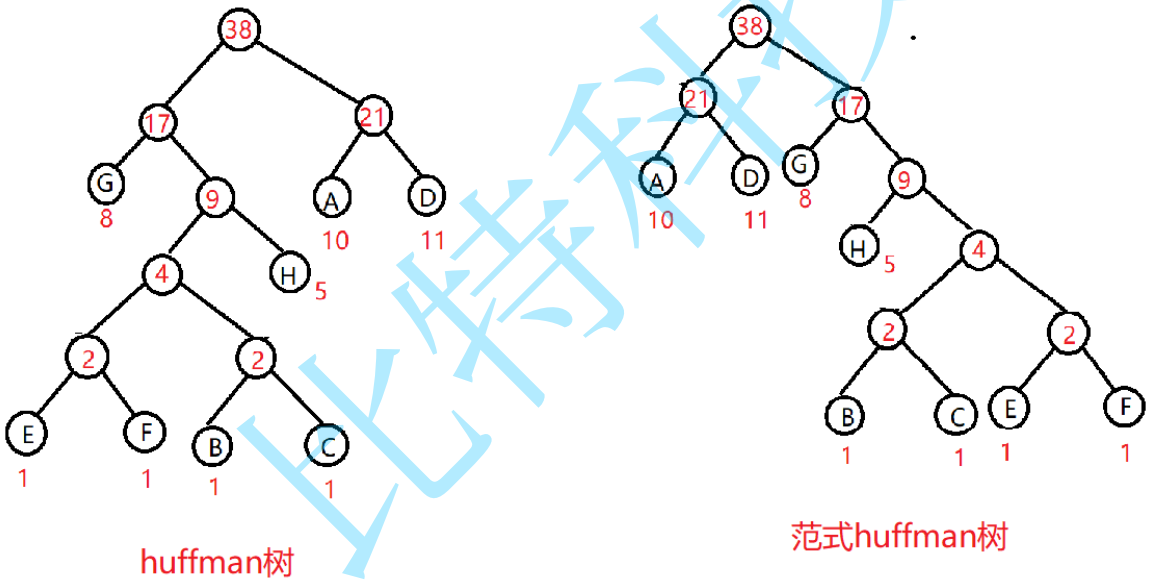
## 9. 范式huffman树

### 9.1 关于huffman树的思考

huffman树压缩完成后，必须要在压缩文件中保存字节频度信息，才可以解压缩，如果字节的频度信息比较大，对压缩率也会有一定的影响。而且解压缩时通过不断遍历还原的huffman树来进行解压，对解压的效率也会有一定的影响，因此在工程中一般很少使用huffman直接进行压缩，而采用范式huffman树来进行压缩。

### 9.2 范式huffman树

**范式huffman树是在huffman树的基础之上**，进行了一些**强制性的约定**，即：**对于同一层节点中，所有的叶子节点都调整到左边，然后，对于同一层的叶子节点按照符号顺序从小到大调整**，最后按照左0右1的方式分配编码。大家仔细观察下图：




| 符号 | 编码位长 | 哈夫曼编码 |
|---|---|---|
| A | 2 | 10 |
| B | 5 | 01010 |
| C | 5 | 01011 |
| D | 2 | 11 |
| E | 5 | 01000 |
| F | 5 | 01001 |
| G | 2 | 00 |
| H | 3 | 011 |

| 序号 | 同组序号 | 符号 | 位长 | 编码 |
|---|---|---|---|---|
| 0 | 0 | A | 2 | 00 |
| 1 | 1 | D | 2 | 01 |
| 2 | 2 | G | 2 | 10 |
| 3 | 0 | H | 3 | 110 |
| 4 | 0 | B | 5 | 11100 |
| 5 | 1 | C | 5 | 11101 |
| 6 | 2 | E | 5 | 11110 |
| 7 | 3 | F | 5 | 11111 |

从上表中可以得出一些结论：

1. 只要知道一个符号的编码位长就可以知道它在范式树上的位置。即：**码表中只要保存每个符号的编码长度**(即节点在树中的高度)即可，其远远要比符号频度小。
2. **相同位长的编码之间都相差1**
3. **第n层的编码可以根据上层算出来：code = (code + count[n-1])<<1**;

范式huffman树该如何创建呢?

范式huffman树根本不用创建，可以利用huffman树推到出来：

1. **对huffman树中的每个叶子节点求层数，得出huffman码表**
2. 对huffman码表按照：**码长(节点在树中的高度)为第一关键字、符号为第二关键字进行排序**

通过以上两步就可以得出范式huffman树的码表，然后按照上面的公式既可以计算出范式huffman码表。

### 9.3 压缩

1. 通过2.2小节中的方式**计算出huffman码表，并推算出每个字符的范式huffman编码**
2. **读取源文件，将源文件中的每个字节按照对应的范式huffman编码进行改写**

压缩文件的格式：

1. **先保存各个字节对应的码字长度**(huffman压缩中保存的是符号及符号出现的频率)
2. **保存压缩数据**

### 9.4 解压缩

1. 从压缩数据中获取符号的编码位长，构建符号位长表

| 符号 | 位长 |
| --- | --- |
| A | 2 |
| B | 5 |
| C | 5 |
| D | 2 |
| E | 5 |
| F | 5 |
| G | 2 |
| H | 3 |

2. 根据编码位长建立解码表。

| 序号 | 编码位长 | 首编码 | 符号数量 | 符号索引 |
| --- | --- | --- | --- | --- |
| 0 | 2 | 00 | 3 | 0 |
| 1 | 3 | 110 | 1 | 3 |
| 2 | 5 | 11100 | 4 | 4 |

编码：可以位长算出来，此处保存成数字 符号数量：通过map或者unordered_map来进行统计 符号索引：在符号位长表中的首次出现下标

3. 解码

注意：范式huffman编码有一个很重要的特性即**长度为i的码字的前j位的数值大于长度为j的码字的数值，其中i > j。**

循环进行一下操作，直到所有的比特流解析完成：设i=0

1. 从解码表的第i行开始，根据编码位长从压缩数据比特流中获取相应长度的比特位。
2. 将读取的数据与首编码相减，假设结果为num
3. 如果num>=符号数量，i++，继续1，如果num小于符号数量，进行4
4. 将符号索引加上num，用该结果从符号位长表对应位置解析出该符号

例如，输入数据“11110”。令i = 0，此时编码位长为2。读取2位的数据“11”与首编码相减等于3。3大于等于符号数量，于是i = i + 1等于1。此时编码位长为3。读取3位的数据“111”与首编码相减等于1。1大于等于符号数量，于是i = i + 1等于2。此时编码位长为5。读取5位的数据“11110”与首编码相减等于2。2小于符号数量，2加符号索引4等于6。从表2.3中可以查到序号为6的符号是“E”。从而解码出符号“E”。跳过当前已经解码的5位数据，可以重新开始解码下一个符号。

## 8. 测试

1. 用不同格式的文件测试程序，对比是否正确(可使用BeyondCompare工具)
2. 测试huffman编码的压缩率
3. 文本文件与二进制格式文件的压缩比率
4. 测试程序的压缩性能
5. 对压缩文件进行二次压缩，多次压缩，给出结果