**创新性探索课题实践报告**

# 选题: 利用概率统计方法实现英语文本压缩

## 小组成员

|  |  |
| --- | --- |
| **姓 名** | **学 号** |
| **窦正** | **22336061** |
| **郑鸿鑫** | **22336313** |

## 摘要

数据压缩一直是计算机科学领域内的重要问题。哈夫曼（Huffman）编码是实现数据无损压缩最常用的变长编码方法之一。然而，传统的哈夫曼编码实现方式存在编码树的存储开销较大、编解码耗时较长、编码信息熵较大等不足，制约了它的应用。本文研究并实现了一种改进的哈夫曼编码方法，该方法不拘泥于传统的树结构，并采用了码长方差最小的范式编码方式，解决了传统编码方法的许多不足。

## 引言

哈夫曼编码是Huffman在20世纪50年代初提出的一种变长编码方法，根据字符出现的概率构造出平均长度最短的编码。哈夫曼编码是一种前缀编码，传统上需要自底向顶构造一棵哈夫曼树来获得字符的编码，并保存这棵树的结构信息以用于解码。然而，保存编码树需要很可观的存储开销，这对实际压缩的效果产生了一定影响。此外，利用树这种较为复杂的结构进行编解码，相较于操作数组这种简单结构，时间、空间复杂度都更高。

目前，已有许多研究从一些不同的角度探索了对传统哈夫曼算法的改进方式。其中，具有代表性的有动态哈夫曼编码[1]、数组哈夫曼编码[2]、范式哈夫曼编码[3]等。仔细考察已有研究后，我们发现，可以综合数组编码和范式编码这两种方案，同时发挥两者的优势，实现一种利用数组实现的范式哈夫曼编码算法。此外，我们还说明了此种编码算法最大限度地减小了码长方差，这一特性可以降低编码的信息熵，在实时信息传输等许多领域带来便利。

## 方法介绍

哈夫曼编码是一种前缀编码，一个字符的编码不能是另一个字符编码的前缀。推而广之，一种编码只要满足前缀编码、码长与传统哈夫曼编码一致这两个条件，即可称其为广义哈夫曼编码。结合文献[2]，我们发现，在满足上述两个条件的前提下，可以构造出性质各异的广义哈夫曼编码。但哪一种编码有最优的性质呢？在查阅大量资料后，我们一致认为，文献[3]中提到的范式哈夫曼编码在编码存储开销、编解码效率等方面性质优异。另外，由于广义哈夫曼编码在前缀编码的基础上仅关心码长，其摆脱了二叉树的限制，通过数组模拟建立二叉树计算码长的方式也可以实现编码算法。

基于上面的讨论，我们利用范式哈夫曼编码，结合数组模拟建树的编码方式，实现了一种既有传统哈夫曼编码的效果，又能最大限度的节省编码信息存储空间，同时还可提高编解码效率的编码方式。另外，在实现的过程中我们还发现，这种编码具有码长方差最小的特点。

### 数组模拟建立二叉树的原理与过程：

使用数组模拟建立二叉树的目的是获得字符编码的码长，即其在传统哈夫曼编码树中对应的深度。

我们首先需要遍历文本，统计每个字符出现的频率。若文本中共出现了n种字符，则新建一个长为2n的无符号整数数组tree，然后将tree[0]初始化为 -1[[1]](#footnote-1)，将tree[1]到tree[n]按字符出现频率的升序依次初始化为各个字符出现的次数。

完成上述准备后，可用下面的方法模拟建立哈夫曼树：

1. 从tree[n+1]到tree[2n-1]，执行：
2. 从tree[1]开始，向后扫描当前位置之前所有没有被步骤iv标记过的位置，找到其中最小值的下标s1和次小值的下标s2；
3. 将当前位置的值更改为tree[s1]与tree[s2]之和；
4. 将当前位置的下标赋值给tree[s1]和tree[s2]；
5. 标记位置s1与s2。
6. 先将tree[2n-1]初始化为0。从tree[2n-2]到tree[1]，执行：
7. 取出当前位置的值k；
8. 将当前位置的值更改为tree[k] + 1。

在这个过程中，循环I模拟建立二叉树，将tree[1]到tree[2n-2]更新为其父节点的下标；而循环II统计各节点的深度，将tree[n]到tree[1]更新为其码长。需要说明的是，在循环I中，父节点的下标值大于孩子节点的下标值，可推出tree[2n-1]为整棵树的根节点。从而，在循环II开始之前将根节点tree[2n-1]的值（即其在树中的深度）初始化为0就不难理解了。

### 范式哈夫曼编码及其降低文本压缩率的原理：

通过上面的讨论，我们已经可以计算出各个字符对应的码长。

我们按照码长的升序进行范式编码。第一个编码为0；从第二个编码开始，如果其码长与上一个相同，则编码加1，如果码长比上一个长，则编码加1后再左移1位。

根据上述规则，只要知道了每个码长对应字符的数量，即可导出所有编码；再按字符出现频率的降序存储所有字符，即可将字符与其编码对应。

此外，这种范式编码还具有一个很好的性质：长度为的码字的前位的数值大于所有长度为的码字的数值，其中。基于这个性质，我们可以构造出范式哈夫曼编码的解码算法，其时间、空间效率较基于树结构的传统算法更高。此算法在文献[3]中有详细说明，这里不再赘述。

综上所述，范式编码向压缩后的文本中存储的编码信息远远少于传统上需要存储树结构的编码方式，降低了文本的压缩率。

## 创新点

### 创新点1：综合应用了数组模拟建树与范式哈夫曼编码

本文给出的算法不必采用指针和结构体来构建哈夫曼树，而且在压缩时不用将整棵树传递到二进制文件中，这节省了内存开销，提高了文件压缩率。此外，该算法生成编码的范式特性使得程序无需将每个字符及其对应的编码都存入二进制文件，而只需传入与码长相关的信息，然后可以根据这些信息来推导字符对应的编码，进一步提升了文件的压缩效果。

### 创新点2：编码码长方差最小

我们给出码长方差的定义：，其中是字符对应的码长，是字符出现的概率，是码长均值。码长方差最小，通俗的讲就是让码长的长度更加平均，让构建出的编码树更加平衡。

下面说明本算法生成的编码码长方差最小的原因：在数组模拟构建二叉树的过程中，循环I的步骤i扫描字符的顺序为从前往后，对应字符的出现频率从高到低，所以程序总是会取相对靠前的最小权和次小权下标。这里相对靠前的意思是，如果某一轮会出现多个相同的最小权或次小权，优先取前面出现的。这种取法的优势尤其体现在这几个相同的最小权或次小权下标分散在n两侧的时候，程序优先取小于n的下标。因为下标小于n的在数组中对应单个字符，即一个孤立的叶子节点，下标大于n的对应建树过程中产生的双亲结点，它一定带着它的子树，如果取大于n的，必然会增加二叉树的深度，不利于二叉树趋于平衡。

我们举一个例子：假定文本为ABBBCCCCCDDDDDDDEEEEEEEEE。根据上文中说明的算法，可推出tree初始化时下标1到5的元素为1 3 5 7 9。循环I的第一轮循环将tree数组更新为6 6 5 7 9 4（其中4为1 + 3的结果，6代表它们的双亲节点下标为6）；第二轮循环将tree数组更新为6 6 7 7 9 7 9（两个带下划线的7代表的是双亲结点的下标）；进入第三轮循环，此时我们发现最小权是7，次小权是9，但是有两个9。在本文的方法中，我们取的是靠前的那个9，它代表的是一个孤立的叶子节点。假设我们取靠后的那个9，会发现它的下标为7，则它存在两个子树为上一轮下划线的那两个节点，甚至第二个下划线的7下标是6，即它也有两棵子树。下面给出所有循环结束后，基于这两种不同选择得到的二叉树：

本文算法得到的二叉树：

1

3

4

5

9

7

9

16

25

取靠后的“9”得到的二叉树：

1

3

4

5

9

7

9

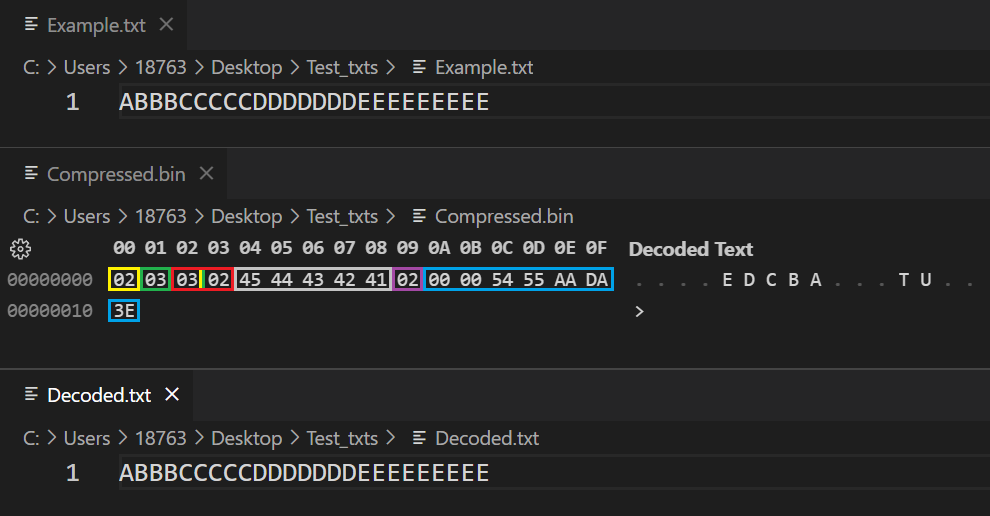
16

25

通过计算得到，本算法生成编码的码长均值为2.16，码长方差为0.1344；取靠后的“9”得到的编码码长均值虽然也是2.16，但码长方差为1.1744。只是一次分支选择的差异，最后会对码长方差造成很大影响。所以我们基于贪心算法，在每次需要做出选择时，总是优先选择靠前的节点，使最后得到的编码码长方差最小。

## 实验结果

### 程序压缩方法介绍：



这里以文本ABBBCCCCCDDDDDDDEEEEEEEEE为例。

上图中，Example.txt为原始文本，Compressed.bin为压缩后的二进制文件，Decoded.txt为解压得到的文本。

00位置上的02表示最短码长为2，01位置上的03表示最长码长为3。

从02位置开始红色框中的部分，按码长升序存储每个码长对应编码的数量。例如这里，码长为2的编码有3种，码长为3的编码有2种；共有5种编码，对应5种字符；由上文中的范式编码规则，可以计算出这5种编码分别为00, 01, 10, 110, 111。

接下来灰色框的位置按字符出现频率的降序存储每种编码对应的字符。例如，这里04~08位置分别存储字符E, D, C, B, A。

由于文件存储的最小单位是字节（8个位），但文本压缩后编码的总长不一定是8的整数倍，可能在文件的最后一个字节产生空余，故需要记录文件最后一个字节有多少个多余的位。之后的一个字节就存储了这个信息，例如，这里的02表示有两位多余。

最后，文件中所有剩余的信息就是文本的哈夫曼编码，这里需要注意，编码按字节的存储方式是倒序的，但按位的存储方式是正序的[[2]](#footnote-2)。例如，这里的二进制信息为00 00 54 55 AA DA 3E，去掉多余的位并转为正常顺序后即得编码0011\_1110\_1101\_1010\_1010\_1010\_0101\_0101\_0101\_0100\_0000\_0000\_0000\_0000。

### 测试用例信息及平均压缩率：

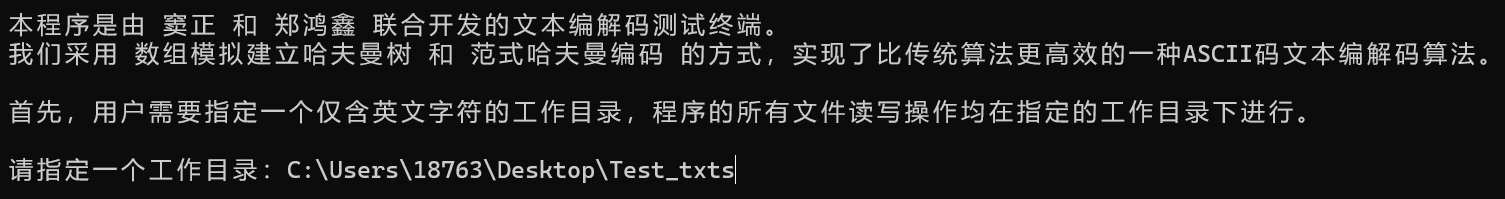
测试文件均为从<https://wiki.ubuntu.com/RandomPage>中保存的随机页面。



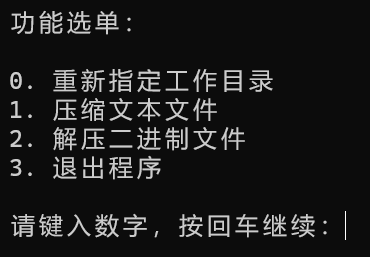
## 程序说明

程序采用C++语言编写，用到了boost/dynamic\_bitset.hpp，编译源代码需要引入并链接boost库。

### 程序开始界面：



### 功能选单：



#### 重新指定工作目录功能：

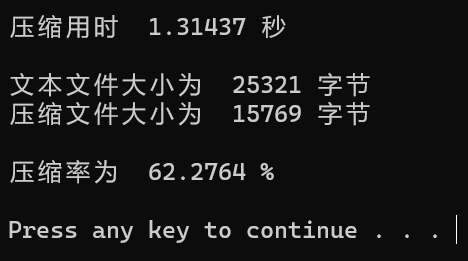


#### 压缩文本文件功能：







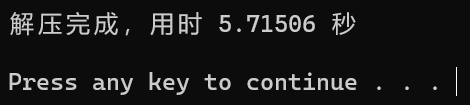


#### 解压二进制文件功能：









## 参考文献

[1] 朱怀宏,吴楠,夏黎春.利用优化哈夫曼编码进行数据压缩的探索[J].微机发展, 2002, 12(5):6.DOI:10.3969/j.issn.1673-629X.2002.05.001.

[2] 王群芳.哈夫曼编码的另一种实现算法[J].合肥师范学院学报, 2006, 24(006):36-38.DOI:10.3969/j.issn.1674-2273.2006.06.013.

[3] 邵天增,尚冬娟.哈夫曼编码应用的一种改进——范式哈夫曼编码[J].科技创新导报, 2008(21):2.DOI:10.3969/j.issn.1674-098X.2008.21.022.

[4] 刘晓锋,吴亚娟.哈夫曼编码的一种基于树型模式匹配的改进型算法[J].西华师范大学学报：自然科学版, 2006, 27(1):5.DOI:10.3969/j.issn.1673-5072.2006.01.016.

## 具体分工说明

### 窦正（22336061）：

我主要负责搜集相关文献资料、起草程序代码架构，设计了二进制文件的存储格式，完成了二进制文件存储、解码这两个功能的函数实现，并设计了测试程序界面，编写了main函数。我编写了本实践报告的摘要、引言、实验结果、程序说明和测试文件介绍部分，并审校了全稿。

本程序需要对文件进行动态位操作，仅利用C++的标准库很难实现需要的功能。为此，我引入了外部库boost中的dynamic\_bitset类，并对照官方文档学习了它的使用方法，基本掌握了读取、存储二进制文件并对其进行位操作的方法。这是我第一次在程序中引入并使用外部库，编程的过程中也遇到了很多挑战，但在我和郑鸿鑫同学的共同努力下，困难被一一击破，最终成功地实现了二进制文件的存储与解码操作。

### 郑鸿鑫（22336313）：

我主要负责阅读文献资料、实现编码算法，完成了编码文本字符和记录编码时间这两个功能的实现。我撰写了本实践报告的方法介绍、创新点两个部分的内容，并与窦正同学一起检测并修正程序后期出现的问题，经过我们的细心排查，修改后的程序对所有的测试样例都可以正常实现编解码功能。

在此次课题实践中，我感受到了查阅文献对学习的巨大帮助，在实践过程中通过文献得到了启发，将数组模拟建树和范式哈夫曼编码进行结合，最终得出了本文的基本编码方法。而且我也体会到在编写程序时要十分严谨，每一步都有可能产生bug，本次实践也提高了我调试和修正程序的能力。

1. -1表示无符号整数所能表示的最大值。 [↑](#footnote-ref-1)
2. 这是由于本程序利用C++ boost库中的dynamic\_bitset类来实现位操作，而此种存储方式是dynamic\_bitset类的特性。 [↑](#footnote-ref-2)