**创新性探索课题实践报告**

# 选题: 基于马尔可夫链的哈夫曼压缩算法实现

## 小组成员

|  |  |
| --- | --- |
| **姓 名** | **学 号** |
| **李海杰** | **19335095** |
| **李治达** | **19335113** |
| **利诗源** | **19335114** |
| **曾杨乐** | **19335012** |

## 摘要

文件压缩应用广泛，压缩算法通过某种特殊的编码方式将数据信息中存在的重复度、冗余度有效地降低，从而达到数据压缩的目的。而各类压缩算法都与《概率论与数理统计》这门课程知识有着千丝万缕的关系，为巩固课程知识、使所学有所用，我们小组调研文献资料寻找基于概率统计的英文文本压缩方法并尝试改进实现。最终实现了**基于马尔可夫链的哈夫曼压缩算法，在特定测试条件下可实现3%的压缩率；实验测试平均压缩率可达35%。**

## 引言

我们小组广泛查阅相关文献资料，寻找基于概率统计的英文文本压缩方法。经过反复比较、几经权衡后，选择了基于哈夫曼算法的一种改进算法——**基于马尔可夫链的哈夫曼压缩算法。**哈夫曼算法操作简单、易于实现、压缩率高，这些特性使得哈夫曼算法应用广泛。但是实际的压缩数据上下文之间往往存在某种联系性，哈夫曼算法把待编码字符进行单独编码，而没有考虑字符之间的相关性，压缩后的数据仍然存在较大的冗余，有待进一步压缩。针对哈夫曼算法没有考虑待压缩数据上下文之间的联系性，**基于马尔可夫链的哈夫曼压缩算法结合上下文的联系性，相较于传统哈夫曼算法有效提高了压缩率**。

## 方法介绍

1. **哈夫曼算法**

哈夫曼编码使用变长编码表对源符号（如文件中的一个字母）进行编码，其中变长编码表是通过一种评估来源符号出现机率的方法得到的，出现机率高的字母使用较短的编码，反之出现机率低的则使用较长的编码，这便使编码之后的字符串的平均长度、期望值降低，从而达到无损压缩数据的目的。哈夫曼算法操作简单、易于实现使得哈夫曼算法应用广泛。但是实际的压缩数据上下文之间往往存在某种联系性，哈夫曼算法把待编码字符进行单独编码，而没有考虑字符之间的相关性，压缩后的数据仍然存在较大的冗余，有待进一步压缩。

1. **马尔可夫链**

马尔可夫链是概率论和数理统计中具有马尔可夫性质且存在于离散的指数集和状态空间内的随机过程。马尔可夫链包含一个有限的状态空间集合，各个状态之间可以以一定的概率相互跳转，概率的大小反映了状态之间的联系性。概率越大说明状态之间的联系性越大，概率越小说明状态之间的联系性越小。

1. **基于马尔可夫链的哈夫曼压缩算法**

针对哈夫曼算法没有考虑待压缩数据上下文之间的联系性，基于马尔可夫链的哈夫曼压缩算法结合上下文的联系性，有效提高了压缩率。为了判断上下文之间的联系性，寻找最大的相关序列，还引入了滑动窗口模型。哈夫曼算法主要包括压缩和解压缩两个步骤，本算法只对压缩步骤进行改进。

1. **马尔可夫链模型**

此算法基于哈夫曼算法的研究，利用马尔可夫链数学模型进行了抽象分析。马尔可夫链包含一个有限的状态空间集合，各个状态之间可以以一定的概率相互跳转，概率的大小反映了状态之间的联系性。概率越大说明状态之间的联系性越大，概率越小说明状态之间的联系性越小。在实际的待压缩数据中，每个字符是由固定长度的字节表示的，因此可以把该字节看做一个马尔可夫链状态。字符之间的跳转可以看做状态之间的跳转。因此，马尔可夫链模型如下所示：

1. *T*是待压缩文本，字符空间为*V={V1,V2,V3……Vn}*，*Vi* 是*V*中的第*i*个字符；
2. *S*是独立编码集合，用来存放独立的编码单元。初始为空，即*S={ }*；
3. *Ｃ*是一维数组，*Ci* 表示字符*Vi* 出现的次数；
4. *Pr*是一阶前向转移矩阵，*Prij* 表示字符*Vi* 前面出现字符*Vj* 的次数；*Ne*是一阶后向转移矩阵，*Neij* 表示字符*Vi* 后面出现字符*Vj* 的次数；
5. *P(n)*是*n*阶前向转移概率矩阵，*N(n)*是*ｎ*阶后向转移概率矩阵。

考虑到在实际的待压缩数据中上下文之间的相关性是一个双向的过程，即不仅要考虑*A*字符在*B*字符之后出现的概率，还要考虑*B*字符在*A*字符之前出现的概率大小。因此该算法在原马尔可夫链的基础上引入了前向转移概率。后向转移概率和前向转移概率的大小共同决定了上下文之间的相关性。

相关性定义：

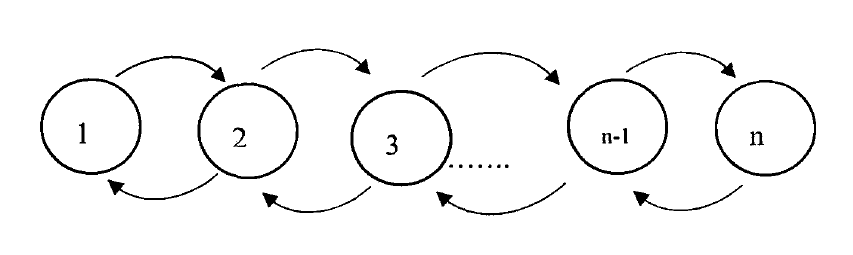
**定义1** 设有字符序列*ViVi+1Vi+2……Vj-1Vj* ，*Vi* 和*Vj* 之间跳转的*n*阶转移概率*Pr(n)ij* 大于*n*阶阈值*TEMP*时，且*Vi* 和*Vj-1、Vj-2……Vi+1* 之间的相应转移概率也分别大于*n-1*阶、*n-2*阶……*1*阶阈值*TEMP*时，则称*Vi* 和*Vj* ***n*阶前向相关**。

**定义2** 设有字符序列*ViVi+1Vi+2……Vj-1Vj* ，*Vi* 和*Vj* 之间跳转的*n*阶转移概率*Ne(n)ij* 大于*n*阶阈值*TEMP*时，且*Vi* 和*Vj-1、Vj-2……Vi+1* 之间的相应转移概率也分别大于*n-1*阶、*n-2*阶……*1*阶阈值*TEMP*时，则称*Vi* 和*Vj* ***n*阶后向相关**。

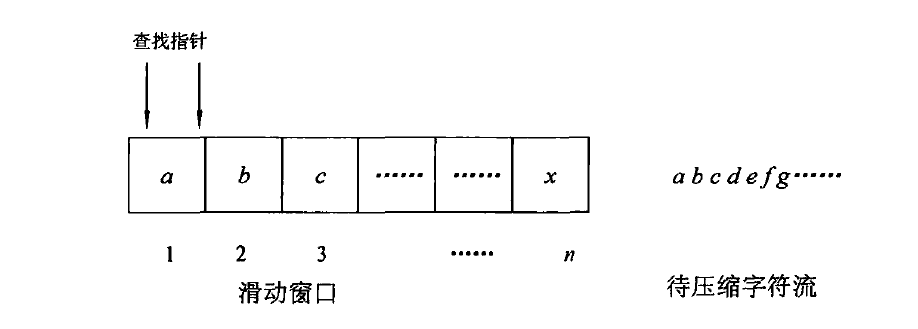
**定义3** 字符*Vi* 和*Vj* 当且仅当满足前向和后向相关时，称*Vi* 和*Vj* **相关**。

阈值*TEMP*的大小的选取将会在下文讨论。

假设有*n*个状态，*n*个状态之间的状态转移图如下：

由定义可知，状态1和状态*n*相关时，状态1和状态2、3……*n-1*也相关。

**3.2 滑动窗口模型**

为了在待压缩数据流中寻找最大的相关序列，此算法采用了滑动窗口模型。滑动窗口大小为*n*，初始状态为空。压缩开始前，滑动窗口位于压缩数据流最左端，随着数据的不断压缩，窗口向右滑动，滑动长度由相关序列的长度决定。滑动窗口模型如下图所示:

为了截取最大的相关序列并对其进行定位，该模型在窗口中设置了两个查找指针，记为x和y，指针与窗口头部的距离称为偏移量。初始时刻，查找指针x和y均位于滑窗口头部，即第一个窗口。

压缩开始时，首先向滑动窗口内读入*n*个长度的字符流，其中x指针先向右移动一格，判断当前窗口内字符与上一窗口内字符的相关性：

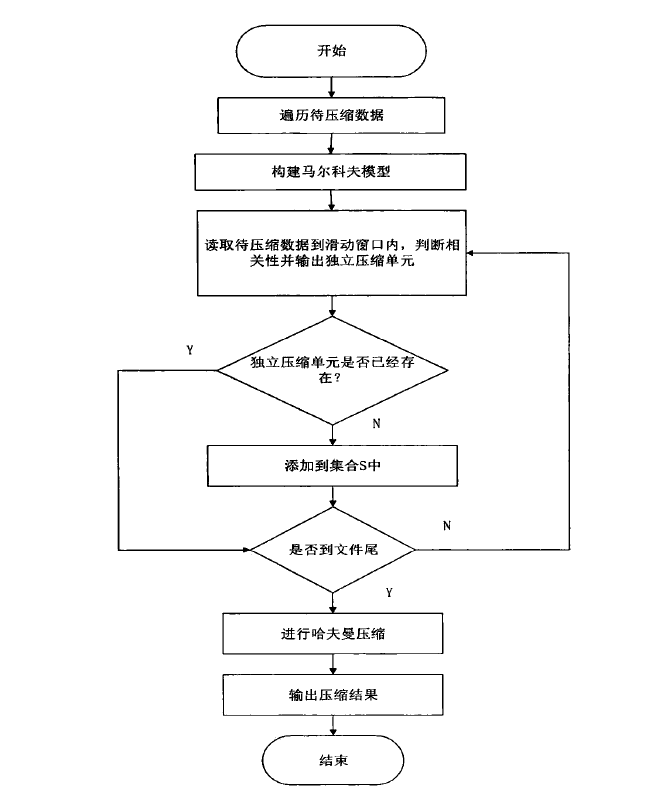
如果不相关，将窗口最左端字符添加到独立编码集合*Ｓ*中去，窗口向右滑动一格，指针重置。如果相关，则x指针继续向后滑动一格重复上述过程，直到找到一个不满足相关性的字符，然后x指针回退一格，假设其偏移量为*j*。这一过程是判断相邻两个字符的相关性，即一阶的相关性。

此时，y指针开始沿窗口向右滑动。第一次移动两格，判断所指窗口内字符与窗口头部内字符的相关性，如果相关，则y指针向右移动一格，重复上述过程直到发现一个不满足相关性的字符，y指针向左回退一格，其偏移量为*i*。这一过程是判断滑动窗口头部字符和其不相邻字符之间的相关性，即*n*阶的相关性。

如果两指针的偏移量相同，即*i=j*，则最大相关序列为窗口头部到第*i*（或*j*）个窗口内的字符序列；如果不同，则最大相关序列为窗口头部到第*i*个窗口内的字符序列。滑动窗口将此序列添加到独立编码集合*Ｓ*中，向右滑动*i*个大小的窗口，指针重置，均指向滑动窗口头部。

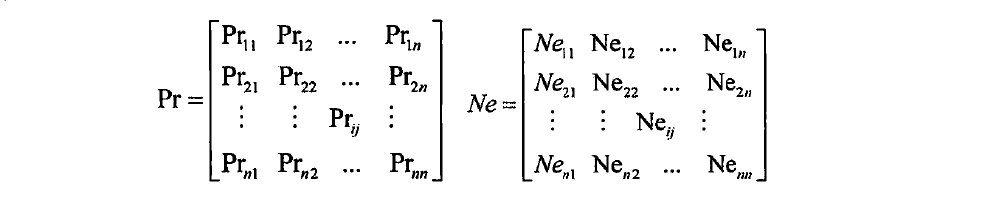
**3.3 算法流程**

基于马尔可夫链的哈夫曼压缩算法如下图所示:



算法的详细描述如下:

**步骤一:** 首先遍历*T*，得到*T*的字符空间 *V={V1,V2,V3……Vn}*，并统计字符*Vi* 出现的次数*ci* 以及字符*Vi* 前面字符*Vj* 出现的次数*Prij* 和*Vi* 后面出现*Vj* 出现的次数*Neij* 。*Prij* 构成一阶前向转移矩阵*Pr*，*Neij* 构成一阶后向转移矩阵*Ne。Pr*和*Ne*如下图所示：

**步骤二：**根据矩阵*Pr*和*Ne*可以得到一阶前向转移概率矩阵*P(1)* 和一阶后

向转移概率矩阵*N(1)*，其中：

矩阵*P(1)*第*i*行第*j*列的元素*pij* 的值为

*pij* = *P(Xn+1=j\Xn=i) = Prij/ik*

矩阵*N(1)*第*i*行第*j*列的元素*nij* 的值为

*nij = P(Xn+1=j\Xn=i) = Neij/ik*

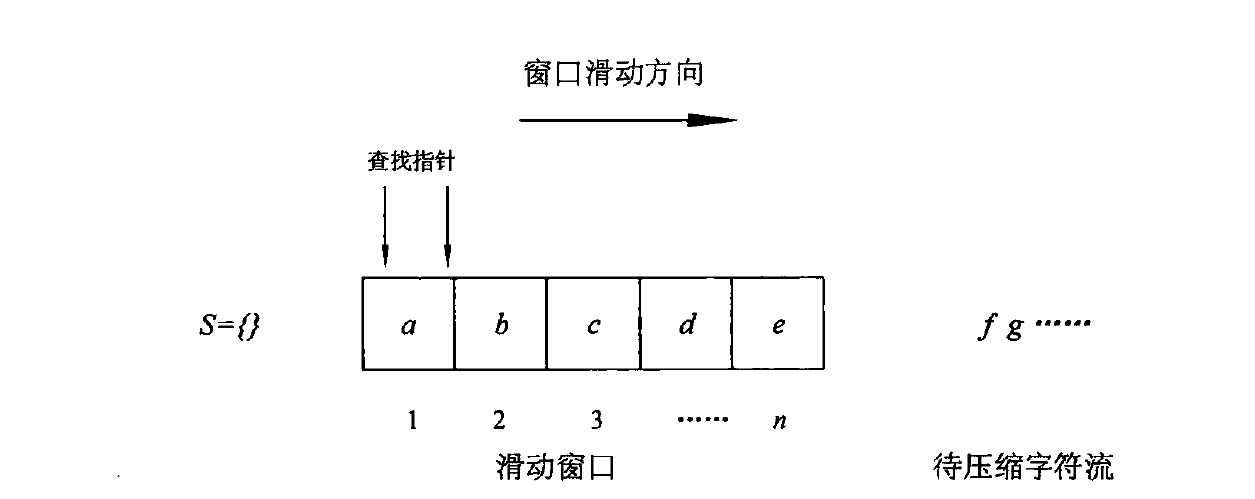
由一阶前向转移概率矩阵和一阶后向转移概率矩阵可得*n*阶前向和后向转

移概率矩阵，即为一阶矩阵的*n*次幂：

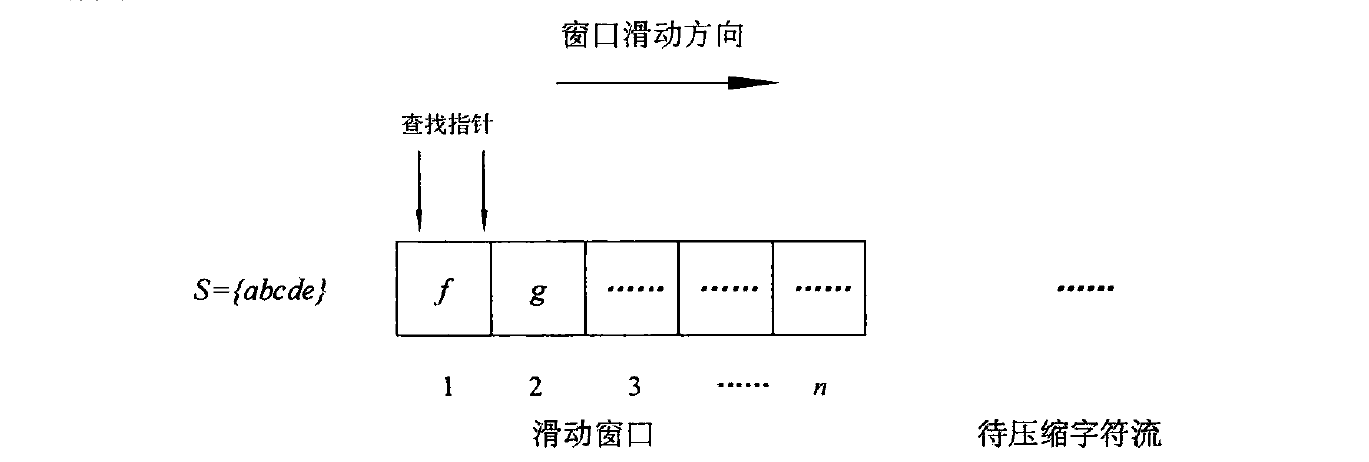
*P(n)= (P(1)) n= P(1)\* P(1)\*……P(1) (n>=2)*

*N(n)=(N(1)) n= N(1)\* N(1)\*……N(1) (n>=2)*

**步骤三：**设置一滑动窗口，设窗口大小为*N*，初始状态为空。读入*N*个字符到滑动窗口内。窗口从文本首部开始向尾部滑动，依次判断窗口内待压缩文本的相关性。首先判断一阶相关性，即相邻字符之间是否相关。如果不相关，则直接输出最左端字符到独立编码集合*S*（若该字符不在S中，则将其次数记为1；若该字符已在S中，则将其次数加1），窗口向右滑动一格；如果相关，则指针右移一个窗口继续判断下一个相邻字符之间的相关性，直到找到一个不相关的字符为止。然后开始判断二阶、三阶……*n*阶相关性，直到找到一个最大的相关序列并输出到*S*中。



以上图为例，首先判断字符*a、b*之间的相关性，如果不相关，则直接输出字符*a*作为独立编码单元。判断集合*S*中是否包含*a*，如果包含则将其次数加1；否则，直接将*a*添加到集合*S*中去，并将其次数记为1。如果相关则判断*b、c*之间的相关性，如果相关则继续判断*c、d*之间的相关性，依次继续下去，直到不满足相关性判断条件为止或到达窗口尾部。然后从窗口头部开始，判断*a、c*之间的相关性，如果相关则继续判断*a、d*之间的相关性，依次进行下去，直到找到一个不满足相关性的字符或查找指针到达窗口尾部；最后将其前面的相关序列输出。如果*a、b、c、d、e*均相关，则输出*abcde*作为一个独立编码单元。判断集合*S*中是否包含*abcde*，如果包含则次数加1，否则把*abcde*添加到*S*中去，并将其次数记为1，然后窗口向右滑动5格，如下图所示：



重复上述过程直到窗口滑动到待压缩文本末尾。

**步骤四：**以集合*S*中独立编码单元出现的次数作为它们的权值*Wi* 。对给定的*n*个权值构成*n*棵二叉树的初始集合*T*，其中每棵二叉树*Ti* 中只有一个权值为*Wi* 的根节点，左右子树均为空。

**步骤五：**在集合*T*中选取两棵根节点权值最小的树作为新构造的二叉树的左右子树，新二叉树的根节点权值为其左右子树的根节点权值之和。从*T*中删除这两棵树，并将新构造的树加入到*T*中。重复该过程，直到集合*T*中只剩下一棵二叉树为止，该二叉树即为待压缩文本的哈夫曼树。根据哈夫曼树构建哈夫曼表。

**步骤六：**设置一滑动窗口，窗口大小为*N*(与步骤三相同)，有头指针和尾指针，分别指向窗口内的第一个字符和最后一个字符。该滑动窗口从待压缩文本的头部向尾部滑动。根据窗口内的文本去查找哈夫曼表：若找到，则输出该文本单元的压缩编码，头指针指向尾指针的下一个字符，重置窗口大小为*N*；若未找到，则窗口的尾指针向前移动一个字符，重新进行查找。重复以上过程，直到所有文本单元都被查找并输出压缩编码，得到压缩后的二进制文件。

**步骤七：**对压缩后的二进制文件进行解压缩。借由对哈夫曼树的遍历，将压缩后的二进制串一步一步还原。解码步骤与经典哈夫曼压缩算法相同，无需在此赘述；有一点需要注意的是，遍历到哈夫曼树叶子节点所代表的文本单元的字符长度取值为[1,N]。将解压缩后的文本输出到新的文件中，与原文本进行比对。

1. **压缩阈值的确定**

19世纪的意大利经济学家*Pareto*研究了个人收入的统计分布，提出个人收入Ｘ不小于某个特定值ｘ的概率与ｘ的常数次幂亦存在简单的反比关系：*P[X>=k]~ x(-k)*上式即为*Pareto*定律，也叫二八定律。统计结果显示，某个字符后面或前面出现其他各字符的次数基本符合*Pareto*定律。

后向转移概率阈值*TEMP*（即后向阈值）和前向转移概率阈值*temp*（即前向阈值）的大小会影响最终的压缩效果，可以通过设置不同的阈值参数来实现最优的压缩效果。根据一阶字符的统计结果，基本符合*Pareto*定律，因此令一阶后向阈值*TEMP(1)* 和一阶前向阈值*temp(1)* 分别满足下式：

*TEMPi(1)* *= MIN{nij(1)} (jk=1 nij(1)= 0.8)*

*tempi(1)* *= MIN{pij(1)} (jk=1 pij(1)= 0.8)*

*TEMPi(1)*表示第*i*个字符的一阶后向转移概率阈值，*tempi(1)*表示第*i*个字符的一阶前向转移概率阈值。由于随着上下文压缩阶数的提高，上下文之间的联系性降低，因此阈值应随之降低。根据文献，对于字符序列*ViVi+1Vi+2……Vj-1Vj，n*阶后向阈值*TEMPi(n)*和*n*阶前向阈值*tempi(n)*的计算方法如下所示：

*TEMPi(n)= TEMPi(1)\* TEMPi+1(1)\*……TEMPj-1(1)\* TEMPj(1)=∏jk=1 TEMPk(1)*

*tempi(n)= tempi(1)\* tempi+1(1)\*……tempj-1(1)\* tempj(1)=∏jk=1 tempk(1)*

1. **上下文阶数的确定**

随着上下文阶数的提高，联系性逐渐减低，因此选择合适的上下文压缩阶数对压缩结果影响较大。当压缩阶数较小时，随着阶数的提高，压缩率逐渐降低；当阶数超过－定值时，压缩率随着压缩阶数的提高逐渐趋于平稳。理论上来说，阶数越高，压缩率越高，所对应的实现难度也就越大，时间复杂度也越高。但是由于信息熵的存在，压缩率不可能无限降低。因此，在具体的实验过程中，我们权衡了各方面的利弊，最终取阶数为8，得到了比较理想的压缩率。

## 实验结果

根据实验要求，本实验以最终的压缩率来判断效果的优劣。首先进行不同算法对同一文件的横向对比，以确定本算法的优越性，再用确定的算法对不同文本进行压缩，确定压缩率的纵向对比，以确定算法的稳定性。

横向对比：

实验测试以给定的百度云中的标准测试文件为输入文件（字符数：），分别采用普通哈夫曼算法，基于一阶马尔科夫链的算法以及基于八阶的马尔科夫链的算法三种实现方式作对照，最终得出了基于八阶马尔科夫链能得到最好压缩率的结论。

压缩率的计算公式：

压缩率=（压缩后文件的大小+哈夫曼表的大小）/压缩前文件的大小

由于压缩后的文件以位为基本存储单位，压缩前的以字节为单位，因此又可将公式进行如下变化：

压缩率=（压缩后位串的长度+哈夫曼表的位数）/（压缩前文件的串长\*8）

结果可视化如下：

8阶马尔科夫链的压缩效果最佳，能达到约32%，比普通的哈夫曼压缩效果好了进一倍，实验代码提交的也是基于8阶马尔科夫链的哈夫曼算法。

最后，对解码所得的文件进行正确性验证，得到与原文件完全相同的新文件，证明算法有效。

纵向对比：

对txt01文件进行压缩（字符数：7041），压缩率为：32.8350%

Txt01为题给的标准文件

对txt02文件进行压缩（字符数：16548），压缩率为：29.6185%

网址为：

https://wiki.ubuntu.com/ArizonaTeam/Meetings/August%2017%2C%202014%20Meeting

对txt03文件进行压缩（字符数：49），压缩率为：89.3382%

网址为：

https://wiki.ubuntu.com/Blindraven

对txt04文件进行压缩（字符数：17290），压缩率为：31.9649%

网址为：<https://wiki.ubuntu.com/UbuntuWeeklyNewsletter/Issue35/It>

以上文件均放在了压缩包中

通过上面的文件测试可知，当文件字符数小时（如txt03.txt），压缩效果并不是特别理想，但是当文件的字符数较多时（如txt01,02,04），压缩效果更加理想。考虑到字符数较少的文本文件一般并没有太大的压缩需求，所以我们认为本算法有较强的稳健性与实用性

本算法优点：压缩率高，实现相对简单。

本算法缺点：时间，空间的开销大。

## 程序说明

本程序采用C++进行编写，程序支持两种命令，一是压缩命令：ziptxt <源文件名> <压缩文件名>，用户输入待压缩文本的文件名称和保存已压缩内容的文件名称，程序对文本进行压缩并将压缩后的内容输出到压缩文件中；二是解压命令：unziptxt <压缩文件名> <解压后文件名>，用户输入待解压文件的文件名称和保存解压后文本的文件名称，程序对压缩文件进行解压缩并将解压后的文本输出到指定文件中。

本程序囊括了：文本读入模块、数据统计模块、相关性判断模块、构建哈夫曼树模块、压缩编码模块、压缩文件写入模块、解码还原模块、压缩率计算模块。

程序内具体函数所对应模块或功能如下表所示：

|  |  |
| --- | --- |
| 函数名称 | 对应模块/功能 |
| readFlie | 文本读入模块 |
| statistics | 数据统计模块 |
| getval | 计算转移概率 |
| calculatetemp1 | 计算一阶阈值矩阵 |
| getforwardtempn | 计算n阶前向阈值矩阵 |
| getbackwardtempn | 计算n阶后向阈值矩阵 |
| judgeRelation | 相关性判断模块 |
| buildTree | 构建哈夫曼树模块 |
| getHuffmanCode | buildTree函数中所需调用的递归函数 |
| encode | 压缩编码模块 |
| writeBinaryFile | 压缩文件写入模块 |
| decode | 解码还原模块 |
| countRate | 压缩率计算模块 |
| readTxtToString | 将压缩文件内容以string形式读出 |
| writeStringToTxt | 将解码还原后的文本内容写入txt文件中 |

## 参考文献

M.A.韦斯.数据结构与算法分析[M].北京：电子工业出版社，2016

A.帕普里斯.概率、随机变量与随机过程[M].西安：西安交通大学出版社，2012

渠开洋.基于改进哈夫曼的上下文数据压缩算法设计与实现[D].北京：北京邮电大学，2017

## 具体分工说明

每位组员分别用200字左右介绍自己所负责的部分，总结自己在所负责的部分中做出的贡献，以及收获与体会。

李海杰（19335095）：查阅检索文献，实现优化算法的的核心代码

在这次小组实践中，我查阅了大量的文献，在众多编码优化算法中，甄选出效果好又易于实现的基于马尔科夫链的哈夫曼算法。并以此为基础，给组员分配各自任务。同时，我主要负责实现代码优化的核心部分，将原本的基本字符——频率表改进为基于马尔科夫链的字符串——频率表，为哈夫曼编码与解码打下基础。

曾杨乐（19335012）：整合组员思路、撰写报告。

在这次小组实践中，我参与了相关选题的讨论，提升了个人的论文阅读能力与自我学习能力，与组员们几经商榷和权衡后最终确定选题。与具体模块任务不同，我需理清算法的基本思路与程序的总体架构。在程序编写过程中与各组员进行协调指导，团队协作精神得以充分体现，团队合作能力与沟通表达能力也有所提升。在撰写报告的过程中，进一步深化了对该算法原理的理解，提升了代码阅读水平与word软件使用能力。

利诗源（19335114）：实现哈夫曼压缩的核心代码

在这次小组实践中，我参与了相关选题的讨论，学习了这篇基于马尔科夫链的哈夫曼编码压缩文本的论文，了解了原理。我负责普通哈夫曼树部分的主要代码，包括通过频率表建立哈夫曼树，通过哈夫曼树得到哈夫曼编码表，将原文通过哈夫曼编码表压缩成01串和将压缩的01串翻译回原文。通过本次小组实践，我深入学习了解了该算法的原理，巩固学习了数学思想，也锻炼了自己的代码能力。

李治达（19335113）：编写文件读写、整合模块代码、测试代码与debug

在这次小组实验中，我阅读了大量关于马尔科夫链与哈夫曼树的相关论文，并学习了许多关于文件读写的内容，对哈夫曼树存取编码表做出了改进，将表的占用空间降低了约80%，使整体压缩率大大提升；在各模块完成后，我负责将所有模块整合起来，并对代码正确性进行检验与debug；在这个过程中我对这个算法的原理有了更深刻的理解，深刻的体会到了概率论与数理统计在算法编写中的重要性。

附：本次实践通过github使用git协助完成，仓库地址：https://github.com/Poet-Li/zip