第十二届数学中国数学建模网络挑战赛

地址：数学中国数学建模网络挑战赛组委会　　网址：www.tzmcm.cn
电话：0471-4969085　　邮编：010021　　Email：service@tzmcm.cn

第十二届“认证杯”数学中国

数学建模网络挑战赛
承　诺　书

我们仔细阅读了第十二届“认证杯”数学中国数学建模网络挑战赛的竞赛规则。

我们完全明白，在竞赛开始后参赛队员不能以任何方式（包括电话、电子邮件、网上咨询等）与队外的任何人（包括指导教师）研究、讨论与赛题有关的问题。

我们知道，抄袭别人的成果是违反竞赛规则的，如果引用别人的成果或其他公开的资料（包括网上查到的资料），必须按照规定的参考文献的表述方式在正文引用处和参考文献中明确列出。

我们郑重承诺，严格遵守竞赛规则，以保证竞赛的公正、公平性。如有违反竞赛规则的行为，我们接受相应处理结果。

我们允许数学中国网站(www.madio.net)公布论文，以供网友之间学习交流，数学中国网站以非商业目的的论文交流不需要提前取得我们的同意。

我们的参赛队号为：

参赛队员　(签名)　：

队员 1：张钰哲

队员 2：谢佳艳

队员 3：胡　俐

参赛队教练员 (签名)：

参赛队伍组别（例如本科组）：本科组

第十二届“认证杯”数学中国

第十二届数学中国数学建模网络挑战赛

地址：数学中国数学建模网络挑战赛组委会　　网址：www.tzmcm.cn
电话：0471-4969085　　邮编：010021　　Email：service@tzmcm.cn

数学建模网络挑战赛
编 号 专 用 页

参赛队伍的参赛队号：（请各个参赛队提前填写好）：
4322

竞赛统一编号（由竞赛组委会送至评委团前编号）：

竞赛评阅编号（由竞赛评委团评阅前进行编号）：

第十二届数学中国数学建模网络挑战赛

地址：数学中国数学建模网络挑战赛组委会　　网址：www.tzmcm.cn
电话：0471-4969085　　邮编：010021　　Email：service@tzmcm.cn

2019 年第十二届“认证杯”数学中国数学建模网络挑战赛第二阶段论文

题　　目　无监督的字典生成模糊匹配分词结构技术

关 键 词　无监督　分词技术　模糊匹配　允许有误差的 KMP 算法

摘　　要:

本文探讨了在待处理文本中，如何找出由于技术限制而发生错误的词长片段。根据录入错误概率分析模型，大型超完备初始模型，KMP 模型，允许有错误值得 KPM 模型，从而快速、准确的找出符合要求的片段。

针对问题，我们可分为三个阶段，阶段一为找出发生错误的候选集，阶段二为根据待处理文本找到符合要求的词长片段，阶段三为将候选集与超完备词典进行匹配。

在第一阶段，我们建立了录入错误概率分析模型。首先根据出现错误的概率，确定错误出现的次数，其次根据对出错情况的分析，确定候选片段词长的区域，从而确定所需的候选集。

在第二阶段，我们建立了大型超完备初始词典模型。首先根据大型超完备词典模型确定的词长阈值（τ_L）和词频阈值（τ_F），然后用 Apriori 算法选出满足长度小于等于τ_L出现频率大于等于τ_F的繁项集，随后将每个词的出现频率用标准化的词频取代，最后用 EM 算法中估计每个词的实际使用频率，经过多次迭次可以确定待处理文本中对应词长小于τ_L且词频大于τ_F的词语，从而构成所需的超完备词典。

在第三阶段，我们建立了 KMP 模型，允许有错误值得 KPM 模型。首先将候选集与超完备词典进行匹配，然后再利用匹配失败后的信息，尽量减少模式串与主串的匹配次数从而达到快速匹配，最后在允许有错误出现的情况下，对错误出现情况的分析，找到所需的片段。

参赛队号：　4322

所选题目：　B　题

参赛密码 _____

（由组委会填写）

第十二届数学中国数学建模网络挑战赛

地址：数学中国数学建模网络挑战赛组委会　　网址：www.tzmcm.cn
电话：0471-4969085　　邮编：010021　　Email：service@tzmcm.cn

英文摘要（选填）

（此摘要非论文必须部分，选填可加分，加分不超过论文总分的 5%）

This article explores how to find out the length of a word that has been wrong due to technical limitations in the text to be processed. According to the input error probability analysis model, the large overcomplete initial model, the KMP model, allows KPM models with incorrect values to quickly and accurately find the fragments that meet the requirements.

For the problem, we can be divided into three stages, one is to find out the candidate set that has the error, the second is to find the word length segment that meets the requirements according to the text to be processed, and the third is to match the candidate set with the overcomplete dictionary.

In the first phase, we established a model for entering the error probability analysis. First, the number of occurrences of the error is determined according to the probability of occurrence of the error, and secondly, the region of the length of the candidate segment is determined based on the analysis of the error condition, thereby determining the candidate set required.

In the second phase, we built a large overcomplete initial dictionary model. Firstly, according to the word length threshold (τ_L) and word frequency threshold (τ_F) determined by the large overcomplete dictionary model, then the Apriori algorithm is used to select the set of items that satisfy the length less than or equal toτ_L and the frequency is greater than or equal toτ_F, and then the appearance of each word The frequency is replaced by the standardized word frequency. Finally, the actual frequency of each word is estimated by the EM algorithm. After multiple times, the words with corresponding word length less than τ_L and word frequency greater than τ_F can be determined to form the required super completeness. dictionary.

In the third phase, we built the KMP model, allowing for errors to be worthy of the KPM model. Firstly, the candidate set is matched with the overcomplete dictionary, and then the information after the matching failure is used to minimize the matching times of the pattern string and the main string to achieve a fast match, and finally, in the case where an error is allowed, the error occurs. Analyze and find the desired fragment.

目录

一、 问题重述

我们现在存在一种未知且无监督的待处理文本，该文本有 30 段且每段文本长度在 5000-8000 个字母之间。而我们的目的是在待处理文本中，如何尽可能多的找出长度为 15 个字母的片段，在寻找的过程中会出现删失错误、插入错误或替换错误，重要的是每个错误只涉及一个字母、独立发生且概率为 1/5，对于一种未知的语言不管多么复杂，用未知的语言与已知知识进行匹配，便可以将问题简单化，设计出有效的数学模型，并可自行编撰算例来验证算法的效果，快速而尽可能多地找到符合要求的字母片段。

二、 模型假设及符号说明

2.1 模型假设

（1）假设未知语言中的二十个字母默认为 A 至 T；
（2）假设每次修改迭代更新的约束条件都保持平衡；
（3）假设最优结构中找出对应的词语都存在超完备词典中；
（4）假设新词发现用规则方式采集候选集，进行人工筛选时不存在人工干预；
（5）假设出现错误时，三种错误出现的概率都是相等；
（6）假设由于概率过小错误值大于三的情况不予考虑。

2.2 符号说明

序号	符号	符号释义
1	V_P	已有词汇表
2	V_T	待处理文本
3	I_f	频繁项集
4	D_L	大型超完备初始词典
5	D_R	精化词典
6	V_S	分段待处理文本
7	τ_L	词长阈值
8	τ_F	词频阈值
9	θ	未知参数
10	$f(x;\theta)$	概率密度函数
11	T	观察变量（文段）
12	S	隐藏变量（分词结构）
13	$\vec{\theta}$	模型参数（词频）

14	$h_{i(S)}$	字典中第 i 个字在 S 中出现的次数
15	C	定义组合
16	ω_i	第 i 个基本元素（字母）
17	$L(\theta)$	似然函数
18	$P(\omega_j\|\omega_i)$	再出现ω_i的条件下出现ω_j的概率
19	$\hat{\theta}$	极大似然估计
20	$n_k(T)$	每次迭代都要更新词ω_k的使用数在所有可能的分词结构分布下的期望
21	I	错误值计数器

三、 问题分析

分词技术属于自然语言理解技术的范畴，是机器翻译的首要环节，所以分词技术显得尤为重要，第一阶段的分词技术的优劣就直接决定了后续机器翻译工作是否能继续进行，有理论依据，效果好，速度快的分词技术成为了机器翻译的首要条件。而本文带来的一种无监督的字典分词结构技术，可以在无任何先经验知识的条件下，快速的从大规模语料中对语料进行分词。并解决了语料中有错误值的模糊匹配。

四、 模型的建立与求解

4.1 建立基本思路

对于一种未知的语言不管多么复杂，所有的破译在本质上是相同的，即将未知的语言与已知知识进行匹配。罗赛塔石碑的故事已经成为传奇：一个刻有古埃及象形文字的石碑，同样的内容还用希腊语和埃及俗语各刻了一遍。当时人们一直对象形文字摸不着头脑，罗斯塔石碑的发现让语言学家可以通过对照希腊语倒推出象形字母的含义。罗赛塔石碑已经成为语言学习的标志性符号，并被引申为解决某难题的关键要领。

因为我们对这种未知的语言没有任何先经验知识，无法让机器直接学习，对待处理文本进行分词。

首先我们对手头已有的 30 段文本，其中每段文本的长度都在 5000-8000 个字母之间，进行词频分析，作出初步统计，结合待处理文本，生成的集合为大型超完备初始词典；由题知，由于技术限制，当我们在记录每个字母时，都可能有五分之一的概率发生错误。错误类型可能为删失错误、插入错误或替换错误，每个错误只涉及一个字母，且每 个错误的发生是独立的，于是我们利用缺失思想的 EM 算法首先根据已经给出的观测

数据即大型超完备词典，估计出模型参数的值；然后再依据上一步估计出的参数值估计缺失数据的值，再根据估计出的缺失数据加上之前已经观测到的数据重新再对参数值进行估计，然后反复迭代，直到最后收敛，得出我们需要的精化词典，配合最优分词结构对待处理文本进行分词。

综上所述，整篇待处理文本的分词结构依赖于词典的构建和最优化分词结构，上述过程可以用图 4-1 表示：

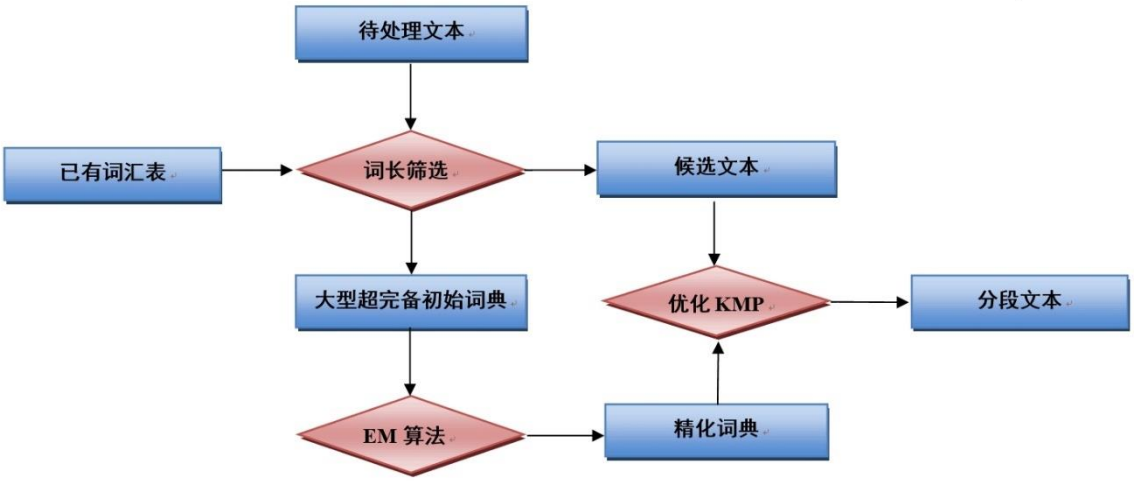


图 4-1-1

4.2 录入错误概率分析模型

首先我们对正确的词长为 15 个字母的片段考虑，由于技术的限制，当我们在记录每个字母时，都可能有五分之一的概率发生错误。错误 类型可能为删失错误、插入错误或替换错误，每个错误只涉及一个字母，且每个错误的发生是独立的。假设在录入这 15 个字母时没有发生一次错误，亦错误数为 0，我们记这种概率为

$$P_0 = C_{15}^{0}\left(\frac{4}{5}\right)^{15}\left(\frac{1}{5}\right)^{0} = 0.0352$$

我们不难得出错误数由 1 到 15 的概率

$P_1 = C_{15}^{1}\left(\frac{4}{5}\right)^{14}\left(\frac{1}{5}\right)^{1} = 0.1319$ ； $P_2 = C_{15}^{2}\left(\frac{4}{5}\right)^{13}\left(\frac{1}{5}\right)^{2} = 0.2309$ ； $P_3 = C_{15}^{3}\left(\frac{4}{5}\right)^{12}\left(\frac{1}{5}\right)^{3} = 0.2501$ ；

$P_4 = C_{15}^{4}\left(\frac{4}{5}\right)^{11}\left(\frac{1}{5}\right)^{4} = 0.1876$ ； $P_5 = C_{15}^{5}\left(\frac{4}{5}\right)^{10}\left(\frac{1}{5}\right)^{5} = 0.1032$ ； $P_6 = C_{15}^{6}\left(\frac{4}{5}\right)^{9}\left(\frac{1}{5}\right)^{6} = 0.0430$ ；

$P_7 = C_{15}^{7}\left(\frac{4}{5}\right)^{8}\left(\frac{1}{5}\right)^{7} = 0.0138$ ； $P_8 = C_{15}^{8}\left(\frac{4}{5}\right)^{7}\left(\frac{1}{5}\right)^{8} = 0.0035$ ； $P_9 = C_{15}^{9}\left(\frac{4}{5}\right)^{6}\left(\frac{1}{5}\right)^{9} = 0.0007$ ；

$P_{10} = C_{15}^{10}\left(\frac{4}{5}\right)^{5}\left(\frac{1}{5}\right)^{10} = 0.0000$； $P_{11} = C_{15}^{11}\left(\frac{4}{5}\right)^{4}\left(\frac{1}{5}\right)^{11} = 0.0000$； $P_{12} = C_{15}^{12}\left(\frac{4}{5}\right)^{3}\left(\frac{1}{5}\right)^{12} = 0.0000$；

$P_{13} = C_{15}^{13}\left(\frac{4}{5}\right)^{2}\left(\frac{1}{5}\right)^{13} = 0.0000$ ； $P_{14} = C_{15}^{14}\left(\frac{4}{5}\right)^{1}\left(\frac{1}{5}\right)^{14} = 0.0000$； $P_{15} = C_{15}^{15}\left(\frac{4}{5}\right)^{0}\left(\frac{1}{5}\right)^{15} = 0.0000.$

（注：为方便数据可视，此处仅取小数点后四位）

将以上数据可视化；

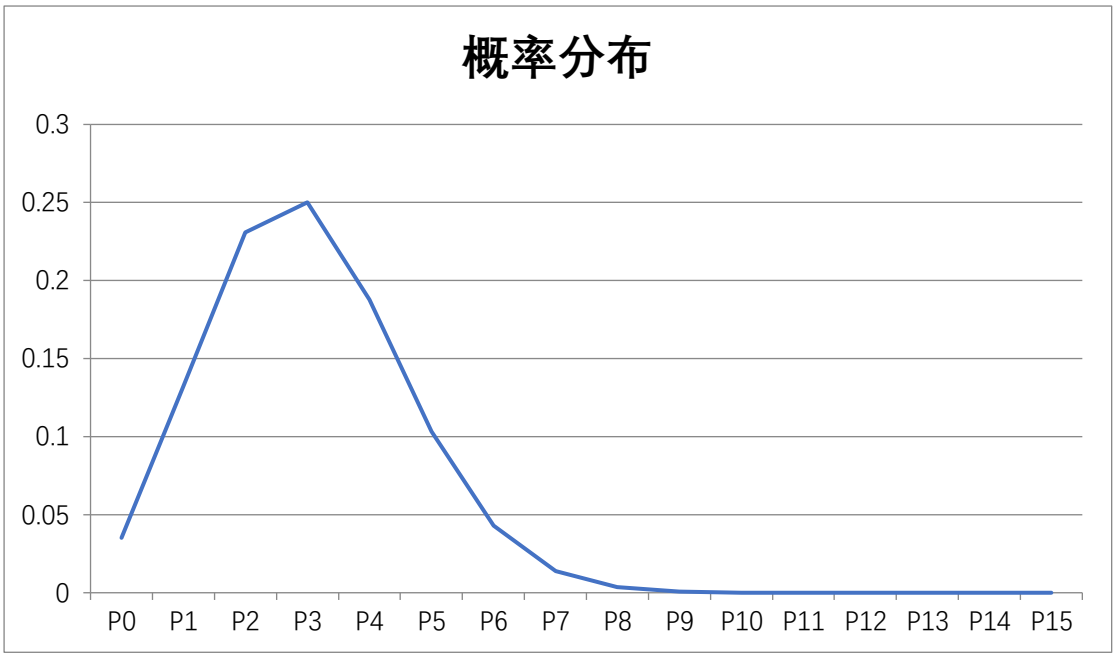


图 4-2-1 错误值的概率统计

不难看出P_8不足1%，在总体概率上看，我们可以认为在录入这 15 个字母的时候错误数最多不超过 7.

接着我们对录入错误的 3 中情况进行分析，其中对词长有影响的只有前两种：删失错误，插入错误。考虑极端情况，7 次录入错误全为删失错误或全为插入错误，计算概率$P_7' = (\frac{1}{3})^7 P_7 = 6.310 \times 10^{-6}$,显然这种概率极小，我们可在这 15 个字母录入的过程中忽略这种情况。同理可得$P_6' = (\frac{1}{3})^6 P_6 = 5.898 \times 10^{-5}$，$P_5' = (\frac{1}{3})^5 P_5 = 4.247 \times 10^{-4}$，$P_4' = (\frac{1}{3})^4 P_4 = 0.002316$，$P_3' = (\frac{1}{3})^3 P_3 = 0.009263$，这里我们近似将错误数为 3 的概率数量级作为百分级，需要注意，此时错误值为 3 意思是在录入 15 个字母字长的片段 3 处错误全为删失错误或全为插入失误。此时对片段字长影响最大，所以可以将有可能的**候选片段词长控制在 12~18 个之间**，换句话说**词长小于 12 或大于 18 的都不可能**。

4.3 大型超完备初始词典模型

我们首先用 C 语言随机生成我们所需要的 30 段文本，并将字母长度控制在 5000~8000 之间，我们进行循环随机生成，第一次生成 5000 个乱码，第二次生成 5100，以此类推最后一次生成 8000 个，其源代码附在附录。

为了判断所生成的片段是否具有可研究性，通过 python 中自带的 difflib 模块，difflib 模块提供的类和方法用来进行序列的差异化比较，它能够比对文件并生成差异结果文本或者 html 格式的差异化比较页面，首先取不同片段进行相似度对比。

这里我们选取第一段比较第二段如：图 2 以及第一段比较第三十段如：图 3。

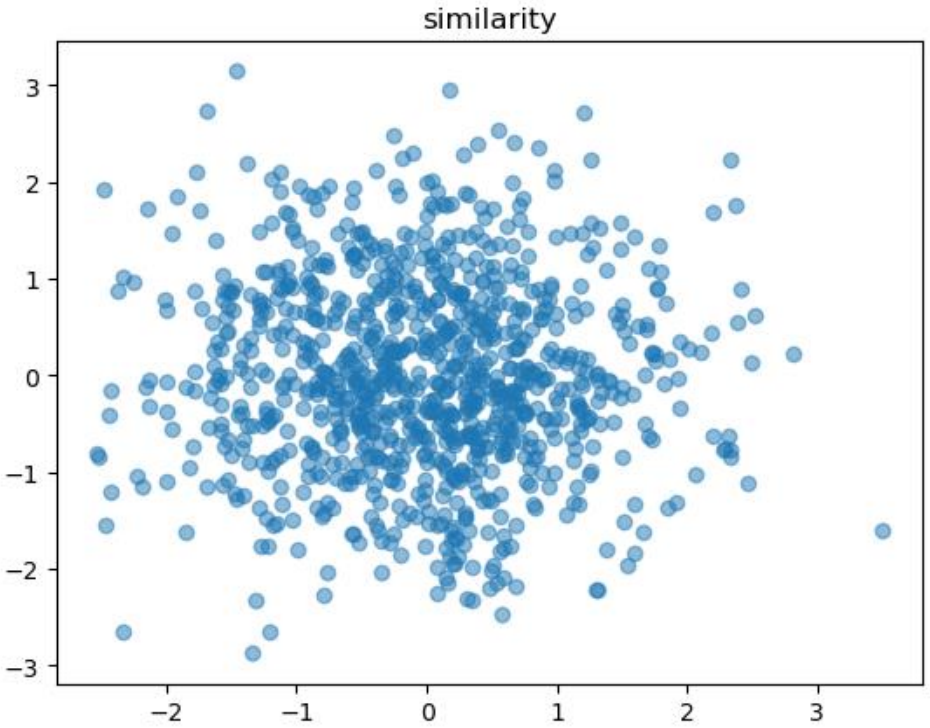


图 4-3-1 第一段与第二段相似度：0.9630

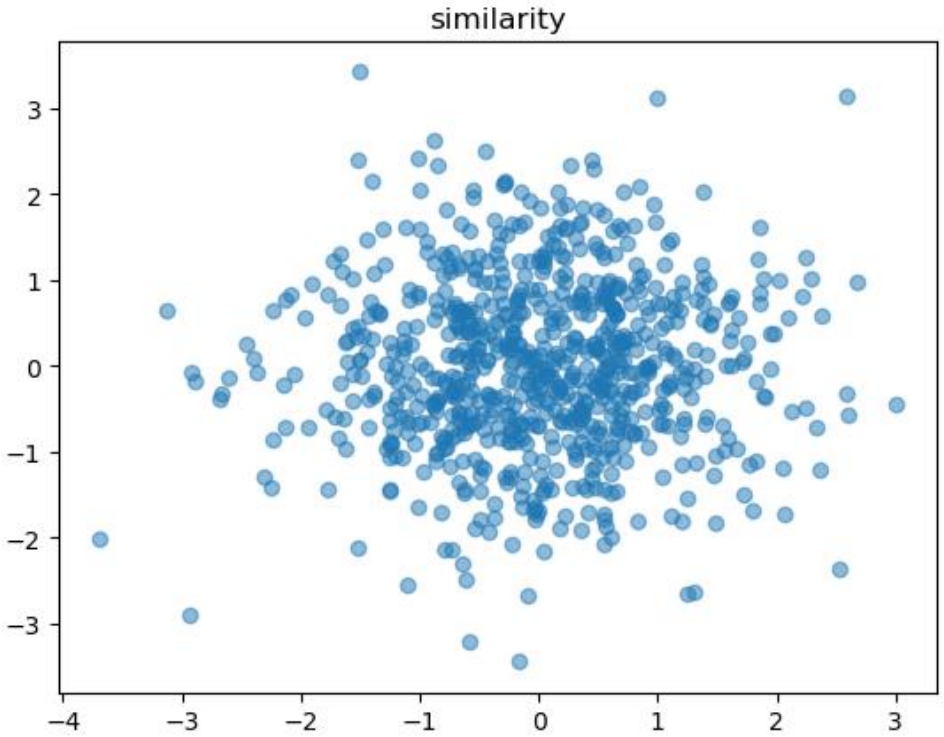


图 4-3-2 第一段与第三十段相似度：0.7752

根据相似系数函数：**两个样本愈相似，则相似系数值愈接近 1；样本点愈不相似，**

则相似系数值愈接近 0。图 1 说明相近的两个片段相似度高，而图 2 即使是片段数量相差最远的也达到了 0.77%的相似度，说明 30 段片段确实可能存在每个文本都出现的序列片段。

4.3.1 用 Apriori 策略构建频繁项集

首先我们对语料进行预处理，企图发现里面存在的频繁出现的组合或者字母与字母之间的关系。发现频繁出现的组合叫做发现频繁项集I_f（frequent item set），而找出字母与字母之间的相关关系叫做关联分析算法。

我们先假设每一条文本片段 T 都是若干个基本元素ω_i的交集。即$t = \{\omega_1 \cap \omega_2 \cap \cdots \cap \omega_k\}$k 是变化的；根据集合的性质可知 t 中的元素都是互斥、无序、确定的。

已有词汇表$V_p = \{t_1, t_2, \cdots, t_{30}\}$，代表着已有 30 段语料的集合。

我们定义组合 C：

$$C = \{\omega_{i1} \cap \omega_{i2} \cap \cdots \cap \omega_{iq} | \forall \omega_{ij}, j \in [1,q], \ni m, m \in [1,n], let\ \omega_{ij} \in t_m\}$$

换句话说，组合就是已有单词表V_p里面若干个字母组合的集合，不难得出每一条语料其实就是一个组合。

在此基础上，我们给出支持度，置信度的概念；一个组合 C 的频率就是组合 C 在已有词汇表V_p上的支持度(support degree),即组合 C 在已有词汇表V_p上的出现概率。

我们继续给出关联规则可信的概念，用条件概率直观解释就是：对于ω_i和ω_j判断其是否关联规则可信，在所有出现ω_i的语料里，有多少处还出现了ω_j。如果这个条件概率特别大，那么我们可以认为“在出现了ω_i的地方一般也会有ω_j”这条规则是可信任的。规则“在出现了ω_j的地方一般也会有ω_i”这条规则是不具备自反性的，也就是说两条规则之间不能等价，因为其本质的条件概率不同：

$$P\{\omega_j|\omega_i\} = \frac{P\{\omega_j \cap \omega_i\}}{P\{\omega_i\}} \neq P\{\omega_i|\omega_j\} = \frac{P\{\omega_i \cap \omega_j\}}{P\{\omega_j\}}$$

同样的，在关联分析中，与某条规则相对应的条件概率也有另一个名称：置信度。

规则$\omega_i \rightarrow \omega_j$的置信度计算公式为:

$$P\{\omega_i \rightarrow \omega_j\} = P\{\omega_j|\omega_i\} = \frac{P\{\omega_i \cap \omega_j\}}{P\{\omega_i\}}$$

生成所有可能的组合方式，然后我们枚举计算这些所有的组合方式的支持度（频率），那些达到支持度指标的就是我们想到找到的频繁项集。很显然，对于每一条语料 T 中的 k 个元素考虑组合数，这 k 个元素可以一个一个组合，也可以两个两个组合……那么这 6 个基本元素组合起来就一共有：$C_k^1 + C_k^2 + \cdots + C_k^k = 2^k - 1$种可能的组合方式，由题设可知$k \in [5000,8000]$，对于这个复杂度可想而知由多困难，需要探查的项集搜索空集可能是指数规模的，于是我们考虑使用 Apriori 原理进行剪枝。

我们先定义一种组合方式，第 i 行，每组 i 个不同元素，向下生成$i+1$个元素的组合，其组合条件为：两组中有且仅有 1 各元素不同的才向下生成；对某一行内元素两两做交集运算，看交集内元素个数是否为$i-1$，例如：

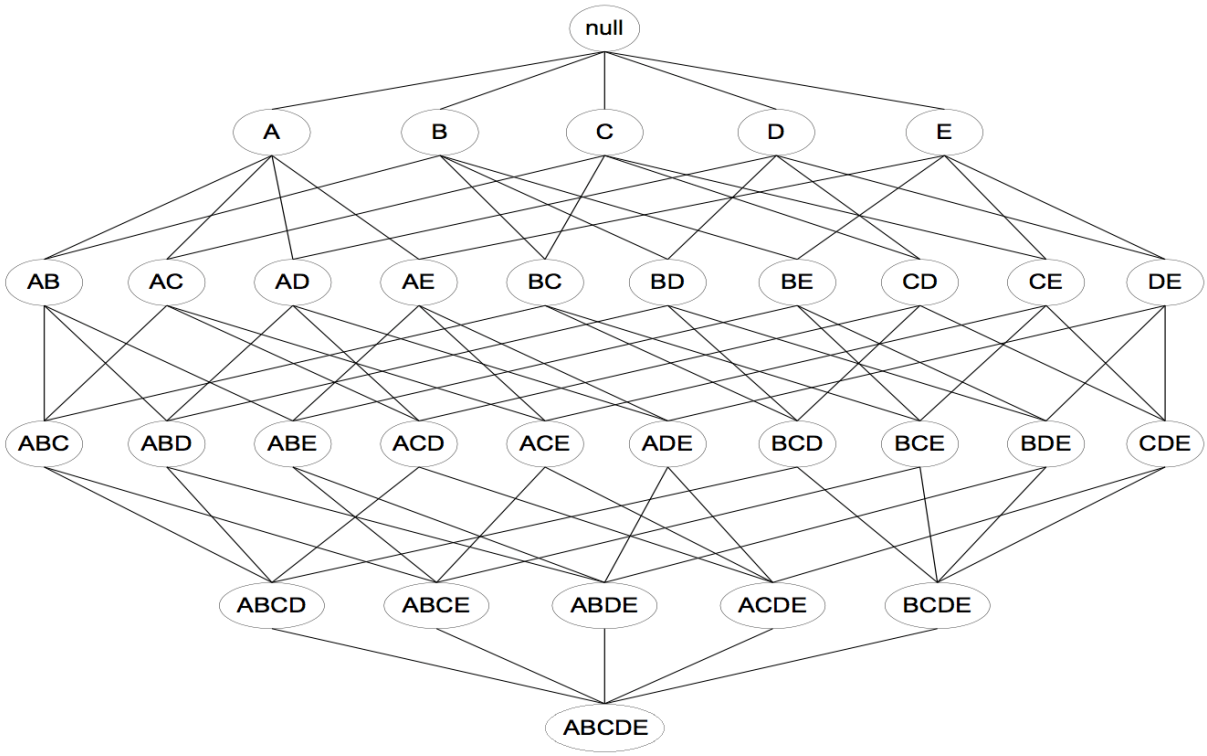


图 4-3-1-1

Apriori 定律 1[2]：如果一个集合是频繁项集，则它的所有子集都是频繁项集。

例如：假设一个集合{A,B}是频繁项集，即 A、B 同时出现在一条记录的次数大于等于最小支持度 min_support，则它的子集{A},{B}出现次数必定大于等于 min_support，即它的子集都是频繁项集。

Apriori 定律 2：如果一个集合不是频繁项集，则它的所有超集都不是频繁项集。

举例：假设集合{A}不是频繁项集，即 A 出现的次数小于 min_support，则它的任何超集，如{A,B}出现的次数必定小于 min_support，因此其超集必定也不是频繁项集。

下图表示当我们发现{A,B}是非频繁集时，就代表所有包含它的超集也是非频繁的，即可以将它们都剪除。

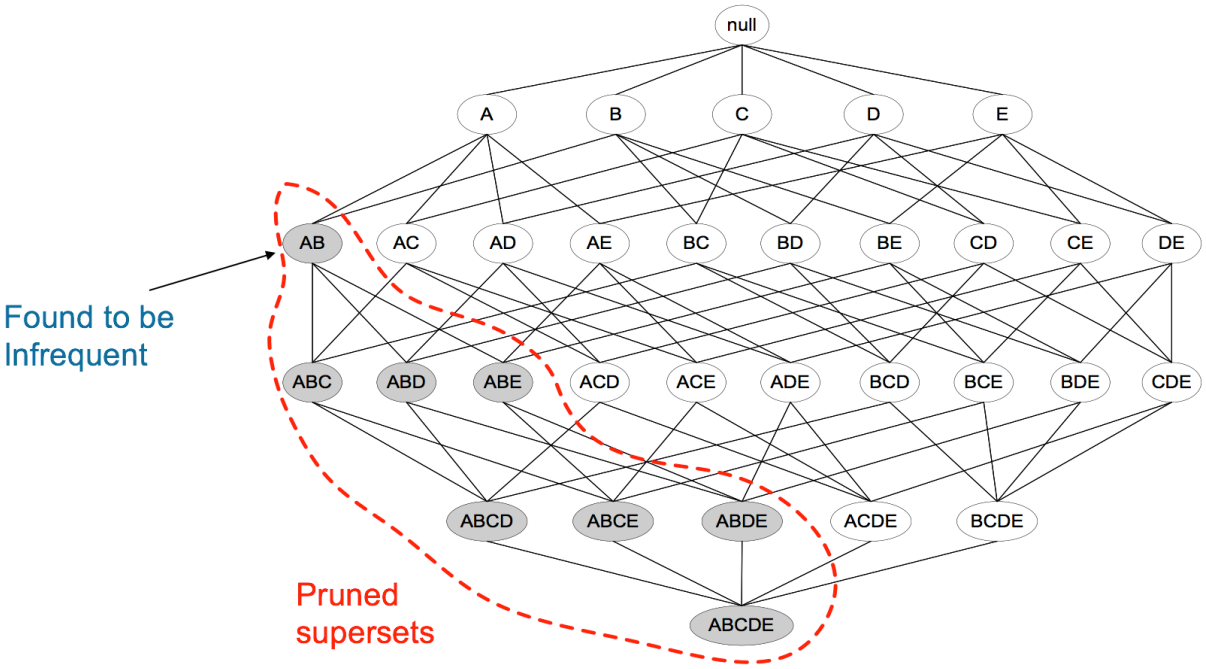


图 4-3-1-2

第一步：生成一部超完备词典。生成方式依赖两个阈值：词长阈值τ_L和词频阈值τ_F。以文本片段为事务，使用上文所述的 Aprori 策略得到长度小于等于τ_L出现频率大于等于τ_F的词，组成超完备词典。

第二步：初始化词频。用标准化后的词频（范围在 0-1 之间）初始化每个词的使用率（word use probability）。

第三步：简化词典。剔除掉概率小于等于 1E-8 的词，的到简化的超完备词典。

4.3.2 优化大型超完备初始词典

首先对于待处理文本V_T在简化的大型超完备词典中进行语料词频检索，对于词频小于等于$1E-8$的词进行剔除。这样就生成了一部专门针对待处理文本V_T的大型超完备初始词典D_L。

4.4 精化词典模型

对于大型超完备初始词典D_L使用 EM 算法得到精化词典模型D_R，采用 EM 算法从语料中估计每个词的实际使用概率，下面是有关符号。

超完备词典：$D_L = \{\omega_1, \omega_2, \cdots, \omega_N\}$

超完备词典的使用率：

$$\vec{\theta} = \{\theta_1, \theta_2, \cdots, \theta \cdots_N\}, s.t. \sum_{i=1}^{N} = 1$$

文本片段：T

T 的一种分词结构：$S = \omega_{i1}, \omega_{i2}, \cdots, \omega_{ik}$

介绍 EM 算法首先需要了解：极大似然估计和 Jensen 不等式。

4.4.1 极大似然估计

X 为连续型，其概率密度函数为$f(x;\theta)$，其中θ为未知参数。设$(X_1, X_2, \cdots, X_n)$是取自总体的样本容量为 n 的简单样本，则$(X_1, X_2, \cdots, X_n)$的联合概率密度函数为$\prod_{i=1}^{n} f(X_i;\theta)$。又设$(X_1, X_2, \cdots, X_n)$的一组观测值为$(x_1, x_2, \cdots, x_n)$，则随机点$(X_1, X_2, \cdots, X_n)$落在点$(x_1, x_2, \cdots, x_n)$的邻边（边长分别为$dx_1, dx_2, \cdots, dx_n$的 n 维立方体）内的概率近似地为

$\prod_{i=1}^{n} f(X_i;\theta)\,dx_i$。

考虑函数

$$L(\theta) = L(x_1, x_2, \cdots, x_n; \theta) = \prod_{i=1}^{n} f(X_i;\theta)$$

同样，$L(\theta)$称为样本的似然函数。

极大似然估计法原理就是固定样本观测值$(x_1, x_2, \cdots, x_n)$，挑选参数θ使

$$L(x_1, x_2, \cdots, x_n; \hat{\theta}) = \max L(x_1, x_2, \cdots, x_n; \theta)$$

这样得到的$\hat{\theta}$与样本值有关，$\hat{\theta}(x_1, x_2, \cdots, x_n)$称为参数$\theta$的极大似然估计值，其相应的统计量$\hat{\theta}(X_1, X_2, \cdots, X_n)$称为$\theta$的极大似然估计量。极大似然估计简记为 MLE 或$\hat{\theta}$。

问题是如何把参数θ的极大似然估计$\hat{\theta}$求出。更多场合是利用$\ln L(\theta)$是$L(\theta)$的增函数，故$\ln L(\theta)$与$L(\theta)$在同一点处达到最大值，于是对似然函数$L(\theta)$取对数，利用微分学知识转化为求解对数似然方程

$$\frac{\partial \ln L(\theta)}{\partial \theta_j} = 0, j = 1,2,\cdots,k$$

解此方程并对解做进一步的判断。但由最值原理，如果最值存在，此方程组求得的驻点即为所求的最值点，就可以很到参数的极大似然估计。极大似然估计法一般属于这种情况，所以可以直接按上述步骤求极大似然估计。

4.4.2Jensen 不等式

（1）定义

设 f 是定义域为实数的函数，如果对于所有的实数 x。如果对于所有的实数 x，f(x)的二次导数大于等于 0，那么 f 是凸函数。Jensen 不等式表述如下：如果 f 是凸函数，X 是随机变量，那么：$E[f(X)] >= f(E[X])$。当且仅当 X 是常量时，上式取等号。

（2）举例

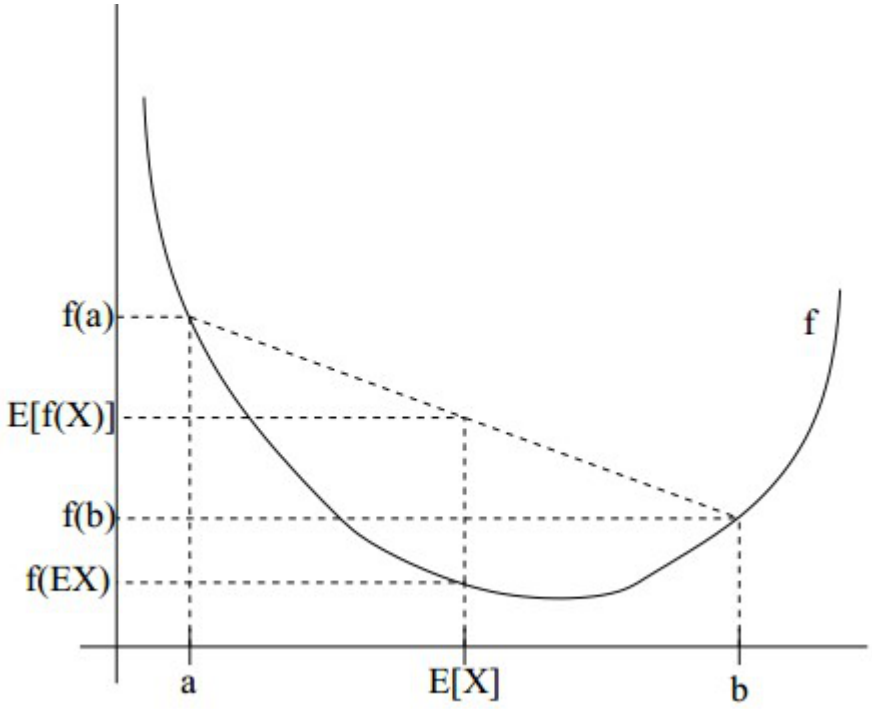


图 4-3-1-3

图中，实线 f 是凸函数，X 是随机变量，有 0.5 的概率是 a，有 0.5 的概率是 b。X

的期望值就是 a 和 b 的中值了，图中可以看到E[f(X)] >= $f(E[X])$成立。Jensen 不等式应用于凹函数时，不等号方向反向。

4.4.3 用 EM 算法得出精化词典模型

推导中采用了 Word Dictionary Model(WDM)模型，即$P(S|\vec{\theta}) = \prod_{k=1}^{k} \theta_{ik}$ 。另外，S 和 T 的关系如下：

$$P(T|\vec{\theta}) = \sum_{S \epsilon C_T} P(S|\vec{\theta})$$

其中C_T表示 T 所有可能的分词结构。

EM 算法的推导

- 观测变量：T(文段)
- 隐藏变量：S(分词结构)
- 模型参数：$\vec{\theta}$(词频)

对于第$r+1$次迭代

$$q(s) = P(S|T, \theta^{(r)}) = \frac{P(S|\theta^{(r)})}{P(T|\theta^{(r)})}$$

Expectation-step：

$$L(\theta; \theta^{(r)}) = \sum_{j=1}^{n} \sum_{s} q(s) \ln P(T, S|\theta)$$

Maximization-step：

$$\theta^{(r+1)} = \arg_{\theta} \max L\left(\theta; \theta^{(r)}\right) \quad s.t. \sum_{i=1}^{N} \theta_i = 1$$

加入约束

$$L((\theta; \theta^{(r)}) = \sum_{j=1}^{n} \sum_{s} q(s) \ln P(T, S|\theta) + \lambda \left(1 - \sum_{i=1}^{N} \theta_i\right)$$

$$= \sum_{j=1}^{n} \sum_{s} q(s) \ln \prod_{i=1}^{i} \theta_i^{h_{i(s)}} + \lambda \left(1 - \sum_{i=1}^{N} \theta_i\right)$$

$$\frac{\partial L(\theta; \theta^{(r)})}{\partial \theta_i} = \sum_{j=1}^{n} \sum_{s} q(s) h_{i(s)} \frac{1}{\theta_i} - \lambda = 0$$

$$\theta_i = \frac{1}{\lambda} \sum_{i=1}^{n} \sum_{s} q(s) h_{i(s)}$$

因为超完备词典的使用率=1

$$\sum_{i=1}^{N} \theta_i = \frac{1}{\lambda} \sum_{i=1}^{N} \sum_{j=1}^{n} \sum_{s} q(s) h_{i(s)} = 1$$

解得：

$$\lambda = \sum_{i=1}^{N}\sum_{j=1}^{n}\sum_{s} q(s)h_{i(s)}$$

$$\theta_i = \frac{\sum_{j=1}^{n}\sum_{s} q(s)h_{i(s)}}{\sum_{i=1}^{N}\sum_{j=1}^{n}\sum_{s} q(s)h_{i(s)}}$$

每次迭代都要更新词ω_k的使用数在所有可能的分词结构分布下的期望。

$$n_k(T) = \frac{\sum_{S\in C_T} n_i(S)P(S|\vec{\theta})}{P(T|\vec{\theta})}$$

动态规划[5]的表示形式

观察发现，$n_k(T)$的分子部分是ω_k在所有可能的分词结构中出现的频率。从另一个角度，文本段所有可能的切分方式来看，可以重新表示成递归的形式。

$$n_k(T) = \frac{\sum_{t=1}^{\tau_L}\theta_{T[1:t]}\cdot P\left(T_{[>t]}|\vec{\theta}\right)\cdot[I(T[1:t]=\omega_k)+n_k(T_{[>t]})]}{\sum_{t=1}^{\tau_L}\theta_{T[1:t]}\cdot P(T_{[>t]}|\vec{\theta})}$$

这样一来，每次迭代更新的是词ω_k的使用数在所有可能的切分分布下的期望。从 T 的尾巴开始就可以动态规划地计算上式。

4.5 基于 KMP 模型的容差模型

为了高效，准确的匹配字符串我们首先是联想到的 KMP 算法，但是 KMP 算法一定要求匹配度达到 100%，所以我们的想法是对 KMP 算法改变一下。

1、若录入时无错误，直接采用 4.5.1KMP 模型。

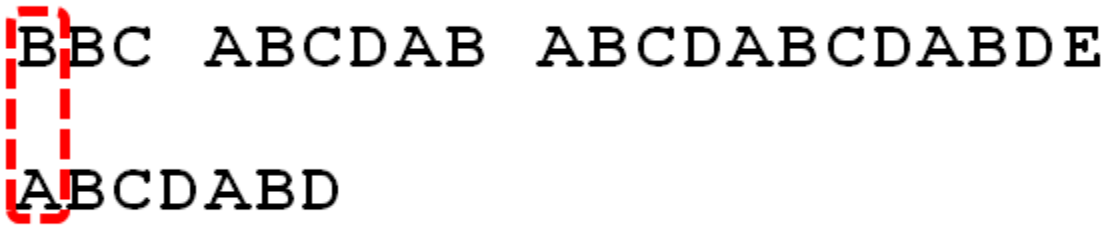
2、若录入有误，则采用 4.5.2 允许有错误值的 KMP 模型。

4.5.1KMP 模型

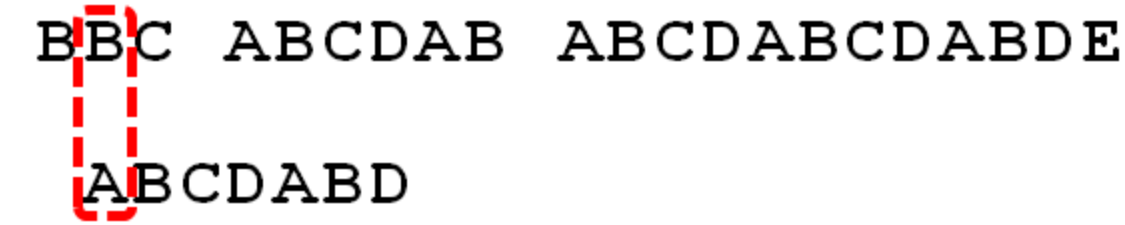
在录入时，如果没有发生错误，即此时为P_0直接用 KMP 算法将其与精化词典进行比较，最后取出符合的片段。

KMP 算法[6]的关键是利用匹配失败后的信息，尽量减少模式串与主串的匹配次数以达到快速匹配的目的。具体实现就是实现一个 next()函数，函数本身包含了模式串的局部匹配信息。

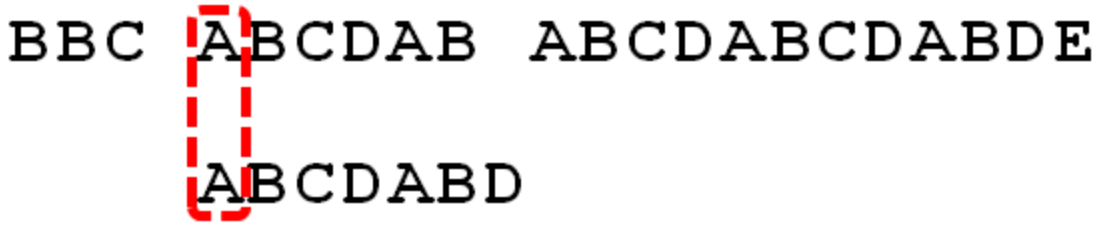
下面我们举具体的例子：



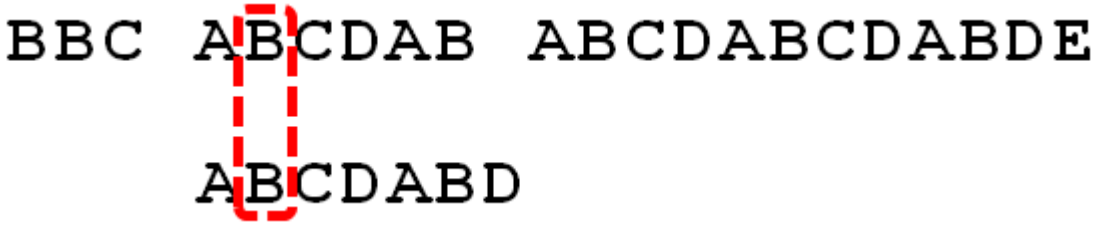
第一步：首先，字符串"BBC ABCDAB ABCDABCDABDE"的第一个字符与搜索词"ABCDABD"的第一个字符，进行比较。因为 B 与 A 不匹配，所以搜索词后移一位。



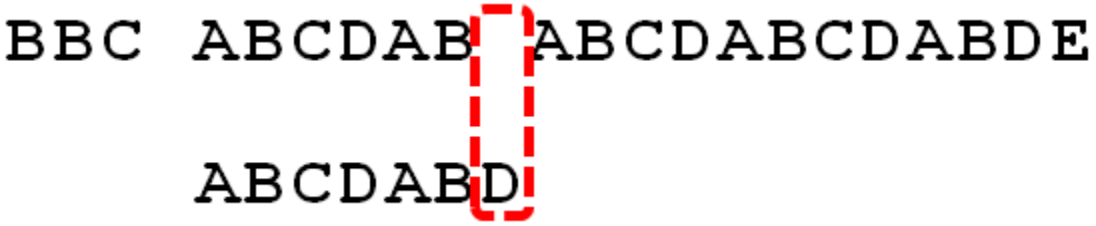
第二步：因为 B 与 A 不匹配，搜索词再往后移。



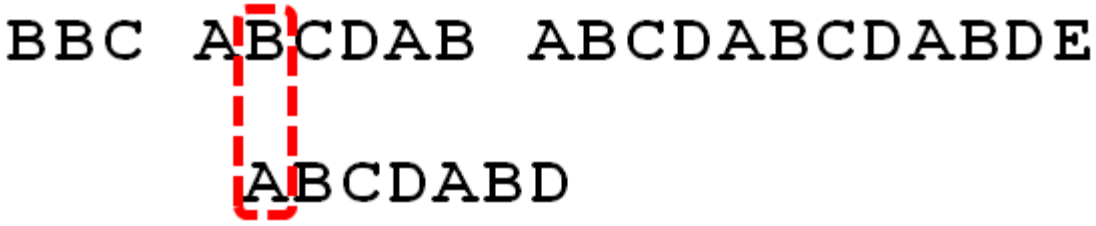
第三步：就这样，直到字符串有一个字符，与搜索词的第一个字符相同为止。



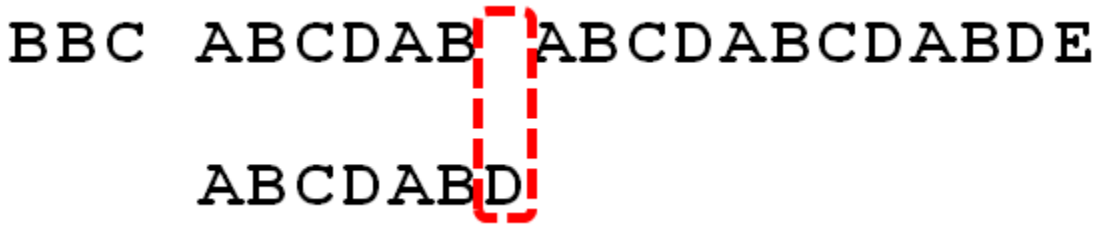
第四步：接着比较字符串和搜索词的下一个字符，还是相同。



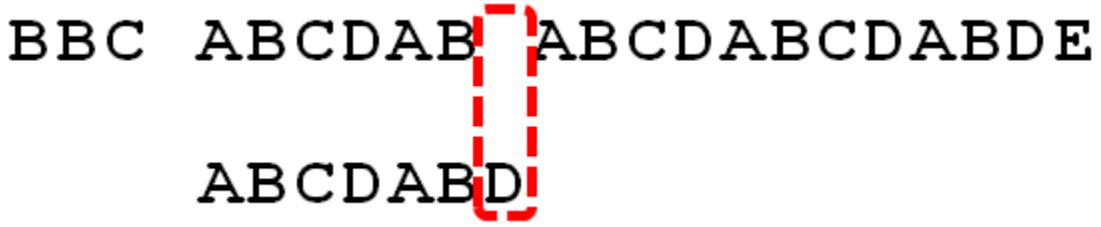
第五步：直到字符串有一个字符，与搜索词对应的字符不相同为止。



第六步：这时，最自然的反应是，将搜索词整个后移一位，再从头逐个比较。这样做虽然可行，但是效率很差，因为你要把"搜索位置"移到已经比较过的位置，重比一遍。



第七步：一个基本事实是，当空格与 D 不匹配时，你其实知道前面六个字符是"ABCDAB"。KMP 算法的想法是，设法利用这个已知信息，不要把"搜索位置"移回已经比较过的位置，继续把它向后移，这样就提高了效率。

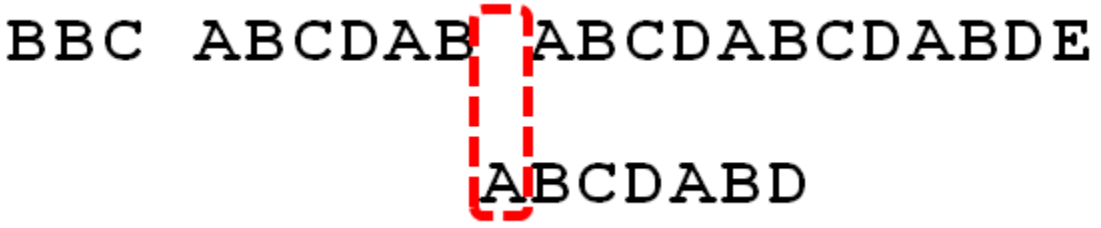


第八步：已知空格与 D 不匹配时，前面六个字符"ABCDAB"是匹配的。查表可知，最后一个匹配字符 B 对应的"部分匹配值"为 2，因此按照下面的公式算出向后移动的位数：

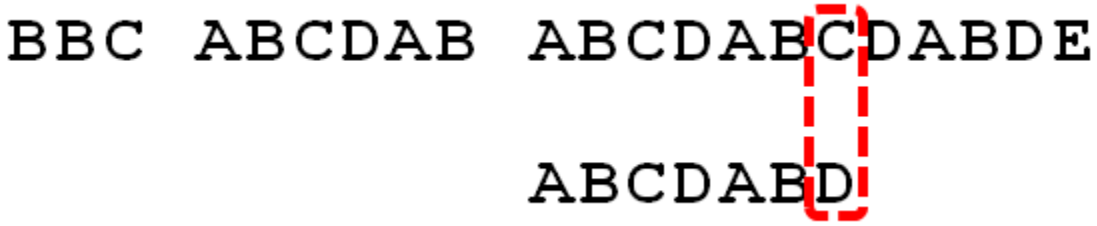
移动位数 = 已匹配的字符数 - 对应的部分匹配值

对已给的算例而言：因为 6 - 2 等于 4，所以将搜索词向后移动 4 位。

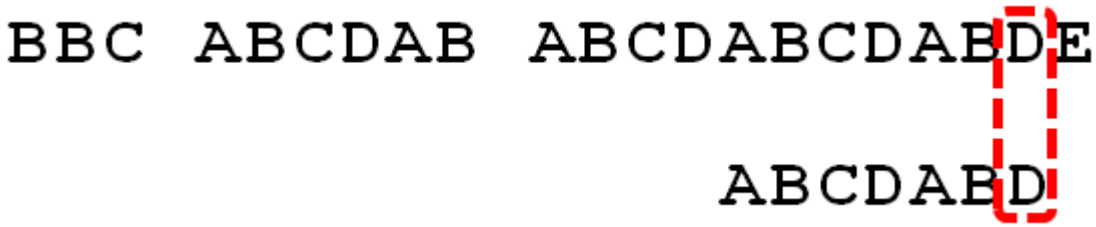
第九步：因为空格与 C 不匹配，搜索词还要继续往后移。这时，已匹配的字符数为 2（"AB"），对应的"部分匹配值"为 0。所以，移动位数 = 2 - 0，结果为 2，于是将搜索词向后移 2 位。



第十步：因为空格与 A 不匹配，继续后移一位。



第十一步：逐位比较，直到发现 C 与 D 不匹配。于是，移动位数 = 6 - 2，继续将搜索词向后移动 4 位。



第十二步：逐位比较，直到搜索词的最后一位，发现完全匹配，于是搜索完成。如果还要继续搜索（即找出全部匹配），移动位数 = 7 - 0，再将搜索词向后移动 7 位，这里就不再重复了。

以上是给出的基本 KMP 算法，当我们录入片段时错误数为 0 时，直接采用 KMP 算法，可以高效准确的取词。其 KMP 算法的缺陷也很明显，就是必须完全匹配才能返回值，取词，而显然我们题目中有容差值不但词长有容差，

4.5.2 允许有错误值的 KMP 模型

对于 3 种错误我们进行以下编号 A：插入错误；B：删失错误；C：替换错误。不难得出在错误值为 3 时，可能出现以下错误组合：

词长	12	13	14	15	16	17	18
组合	BBB	CBB	ABB	CCC	ACC	CAA	AAA
			BCC	ABC	BAA		

表 4-5-2-1

这里对包含三种不同错误的 ABC 组合进行算法分析：

1、对第一位字母进行匹配，若匹配则匹配下一位；若不匹配，$I = I + 1$(I 为错误值计数器)。

2、继续匹配，若匹配则匹配下一位，若不匹配则将字典里的值复制并插入在字符串中，继续匹配，$I = I + 1$，直到匹配成功。若不成功将字符串恢复至步骤 1，进行步骤 3.

3、将字符串中不匹配的字母删除，$I = I + 1$，直到匹配成功。若不成功先恢复再进行步骤 1.

4、$I > 4$时匹配失败。

错误值为 2.

词长	13	14	15	16	17
组合	BB	BC	AB	AC	AA
			CC		

表 4-5-2-2

错误值为 1.

词长	14	15	16
组合	B	C	A

表 4-5-2-3

不难发现以上的 ABC 已经包含了上述 19 种错误排列，当错误值为 2 时仅需改变条件$I > 2$;当错误值为 1 时改变条件$I > 1$.

综上对应不同错误值仅需将判断条件改为I > 错误值.

4.6 对算例的实现

['RILQONTPGRFEPIO' , 'OLPCHHHISLNSELD' , 'OFOCTITPICBSDJ', 'PSKRCCOQJFKFKIA', 'FLRCITOFHMTFAST' ,'PGRAISLMSRRFOHQ' , 'HIPCQNDADORA' , 'SFRGESDXCBCOAN' , 'NCKQQSJJIGKANB GAO' , 'QSRFSGSDCXFLSNF' , 'OFPRPICRKAAOQ' , 'QOJLQH CETQPHKRTMN' , 'TNSKOROIKOGGFQIF' , 'GPDSDATOKLDJRF J' , 'FBTNCFJSKLOJSB' , 'JMDKFDKCQLFC' , 'QQJCSTIKMMC AH', 'BGRKRNLTEBNA' , 'JDRPEMAEAIHEBMA' , 'RGEEBGCES IGAR' , 'HTBNLIKMTDMLGDC' , 'SQJPMGFMHFBDMC' , 'TKQ LJPHNGSGJETSABS' , 'JHEOCTCDCHKCR' , 'JSDDAMRGJDAREIF JBI' , 'LSQBSNMOCPNNIBRH' , 'MHILPSABLSHCJGR' , 'NFGQB MLKBGSKKCM' , 'JJBLCKORHJIGM' , 'EKTMGQACDDFGIKHKRF' , 'IMSKALGBQNMJPCOQ', 'QFCCBJOPSBDCFEP' , 'CIKFQJGBQSO MRTEHDB' , 'NPHJAMMMGJGKBDK' , 'RRMKNNJGTAQBMQFI' , 'PBALIEJHFSEADGL' , 'AQQNRBOHJPKIA' , 'TSQTINFEOOGAPJ A' , 'EJIBGALISTOSFPA' , 'POFSCSEAIJPNJSL' , 'RONAFDBKHPN APRE', 'OJRNGRSBHNFSFCP' , 'SCNGMSDNSJLKGEG' , 'CNPAPGA KQABCKA' , 'MEDOFLGPSJGNOL' , 'CEAOKLCABORQRTB' , 'PLA PATDBHMRHD' , 'KPGEPGAFORHQRD' , 'CDAMCQCAOQOKIIMFD' , 'GQENNBPLNDQDAGLN' , 'PNFSFTBPLSRLKJP' , 'MJNDLNESIOG SBLP' , 'RTOOBDDBOLIDNH' , 'DSBOFBDGLFNPQHJ' , 'NBEADED TIBKCSCDP' , 'KOBGCECLFDEEPMTF' , 'OJASRAIMKRQCHJHA' , 'CTSKSMEJJPNOAOFDP' , 'IFMIHFELPDOCTBLL' , 'BGTINMQOK BFQOS' , 'JLARESNNNCOLMMDRTK' , 'TCBJTFJRACDFOLE' , ' KJOBCLKPKBNSETE' , 'MNMNCDCIGFJCSJ']

五、 模型的评价与推广

5.1 模型的优点与缺点

模型优点：

（1）本文建立的大型超完备初始词典模型无需先天有优势的大产品，思维缜密，完全无监督。

（2）录入错误概率分析模型，大型超完备初始模型，KMP 模型，允许有错误值得 KPM 模型四大模型的灵活运用、层层紧密相连，并且快速且尽可能多的找出符合要求的词语片段。

（3）文中提出的该匹配方法可以快速并且准确找出符合要求的片段。

模型缺点：

（1）新词发现中用规则方式采集候选集，加以人工筛选的方法效果较差，并且需要它们都需要大量的人工干预。

（2）KMP 算法的缺陷也很明显，就是必须完全匹配才能返回值，取词，而显然我们

题目中有容差值且词长有容差。

（3）由于出现错误的次数有很多种，但错误值大于三的情况出现概率太低，不予考虑。

（4）在寻找的过程中，我们将三种错误出现的概率都看成是相等，当概率不相等时便会出现一些误差，无法对错误进行偏好选择。

5.2 模型的推广

对于一种未知的语言不管多么复杂，所有的文字破译在本质上是相同的，用未知的语言与已知知识进行匹配，利用匹配失败后的信息，尽量减少模式串与主串的匹配次数从而达到快速匹配，最后在允许有错误出现的情况下，对错误出现情况的分析，找到所需的片段。

我们可以考虑推广到医学方面上。我们也知道致命的疾病传播速度非常快，假设病人和病毒的基因由小写构成，在收集数据时，会出现很多错误，所以将需要的将患者的 DNA 与病毒的 RNA 进行匹配，用此模型做出高效的病毒检测器，找到一种快速可靠的方法来检测病人体内病毒基因的“脚印”。

六、 参考文献

1: https://blog.csdn.net/jmh1996/article/details/78250556
2: https://blog.csdn.net/baimafujinji/article/details/53456931
3:https://baike.baidu.com/item/%E6%9E%81%E5%A4%A7%E4%BC%BC%E7%84%B6%E4%BC%B0%E8%AE%A1/3350286?fr=aladdin
4:https://baike.baidu.com/item/%E7%90%B4%E7%94%9F%E4%B8%8D%E7%AD%89%E5%BC%8F/397409?fr=aladdin
5:https://baike.baidu.com/item/%E5%8A%A8%E6%80%81%E8%A7%84%E5%88%92/529408?fr=aladdin
6:https://baike.baidu.com/item/kmp%E7%AE%97%E6%B3%95/10951804?fr=aladdin

附录

1、字母乱码程序（C 语言）

```
#include<stdlib.h>
#include<stdio.h>
#include<time.h>
const int min=5000;
const int max=8000;
const int add=100;
int main(void)
{
    int a=0;
    int num=5000;      //每次增加 100
    int n1=min;
    int ns=1;
    char ch='A';


    for(n1;n1<=max;n1+=100,ns++)
    {
       printf("第 %d 次乱码:共 %d 个乱码\n",ns,n1);
       srand(time(NULL));
       for(int n=0;n<=num;n++)
       {
         a=rand();
          ch='A'+a%20;
          printf("%c",ch);
          if(n%100==0 && n!=0)
          printf("\n");
       }
       printf("\n\n");
    }

    return 0;
}
```

此处仅取第一次生成的 5000 个乱码展示：

```
HHTEAQBJPSJSFOGMTAIIBOGOFOTSIPRANOOANOOFDRFBRRHFDLIHGFLJTSSQFSAFRDKPBLORGKI
BJTFFMGKJSBDJJMAALNAFNIHFK
BPGEDHOOFQHKLFCPDBBNNKRMEAIPFOPRNQTJCCRCGAAFIKSLTKJRMDTBOIENMKIHDLFJMQBSAHP
NOAOIFNKQHEHKBGPFIIQHKASR
FSCHRIISBAMEADLFARAEPLROQQQQMJNLNLFHHJDJRJTNCPEOIBHNBGIAFRHQLHPMIEBETLNANEM
RBGCCPIGTTBKSPQBSARGJNMCA
MRRQTJBPOAQBJAPQJIGOCECGPQBMNQNNBDDCRHBLBAKDOGFSSANPFOTMIHETAMNBBHAIIPFNNCA
ERFRMPBBJGGTIGEQEKPCGJLPM
EEINAIDADOOGNONITLTSDDIBPRFPCPEEETGLMHOHAIQHBLCBJFCOFFEMKPRENQQKFPHTSCSRIFE
JOSKGTJAQJKMHANSNRNSNQRIF
CLNTLRAPLIEAHTJITFNFNHMOEQCRLNACMOBDQCTFHIIIHKSPCLTCBDAHQCEKCKQADQSSRJCAPML
MRHCLIDQKMIFITNBAPECIEJKR
```

IMTCHAARTSNFTKORJEENCNETNEEHAMTGSLRMHRHICBMSQMETOQMTMCSTHCKHAGFPGSACLLJENDG
QPASIJDFMDCSMORSGLKGDOINN
SFNEGKLRDCREDJLFJIMGKSKPEJININDHJQTSDTAMCPATLMDTEGMNGNBFKGTTBSCBNLABJNQRGPI
SHNQQFFHGMFAOSCNENLDATTDN
MFHPIJHPNTOODIADKCKNIFQEQHBDKHQCAKCMQCHILIOPJHDNRJTKIGDTSTIJRQLQEJPLKFAHPQO
BCMJNBKINKOLRKKFMHGNDKBJA
RPQKJANTHJNGGQLOJESCRRHLFEJLAPNHKAFLLMQFNSHPAAHBKFKCTHKOLTMETPRAKDHMTTJDBEA
CKNFCBADFINHFRSAGDEQPHNMA
PKLRGGPGFEHRGOEQADIMGODJPDOBJBSTJAHPGSEDBFIPFCRBCQQDMORONDQMOMAISSAJLNLIGTQ
CJFGKMNAKFLRPDNRFQFDJRQGP
HFFHASQTIHQNARKFKRDOSQKBERSBQJJFLSRDKPHPIHKEPFRMLDDBGFNOTLBQRRKLOAAERSJQSGA
NEERPTNJKOEQKSEMLJSIKGNBT
SGHMFLDSRPPPMFCEMMOPMAQAIRHANQQOGAACTLONQNKOMQKSKJMTTSEAOBQORTBGQIQJGOJASLS
KQJTCNIBNFLRDKDORTSRPPNAP
QPISHJARIGLDFNJRMKNSKOBLOLHKSLHLGKAPLSPNMPJNSIFHRQGPIMGGBLPBSTNRTMQTOHTIPNK
EDDMHLEEMFBQRLRKKHBCFOFRK
LDRQCDFISRGNEPNTJSLQFSLPFQHDSASILHMQOHGBRELNGRTKTLEKIBGSPBHHFCTRKPMQAKFLBIE
LGNNQNHORFQICTLLNNSAKQKKA
DDPGSDELIHEMFDJQPDIKTLJNOJGFRQEEKDPHLKMNIJDIMSBTEPGIAJAIKDMPICFMEFRMSACJOEK
TSJHGCAKDKABAMTSFKQKASDKH
BOAEBLGFLICNPFRPIECCLBJJLBBMIRBOBLKQSIGSCSTREHSEQTSDAGQFLKNACEODCRSPBHCCMPA
MEFGKIOFRDKTLGCCFAACIAAGL
SFSRGALHRMIJFAEGMJTRIOBRBAQDCBKSRCDDBFEDBEASTGOASJTGESJJORMDRAAHRRQRQJSOGMS
HSTDBHJPJTJGKMCHPBMCFPCJJ
KNJSBMPJBNHMNPQEJDRKIMQBOODKMHLEGBBFJOKFCQTJJLRGGGFDBOHQMFABRGFDRQKJJHLNTFC
ATOKDBLOQGBMIPSIORLHMTSFI
AOLHFIBJGEDDIFNMOBQRCQLFPPSNEAGMPAJBTEHMAQRKNSJBKNSLKEROKBREQCHMHIBNTSAFDTD
ENKGCCOHPLGHJDKMDKHOMFAMK
TREEMQCCIOIPOTGTRRQOOOIFDASQEENTJSMNDEIGCKQEJRLILOJKPQQOHMIIPRMCPBHATPHRGRJ
NJTSKLERDFJIRCEDQBLKORNSC
CMLEFTEGECEODMAGKAFIIPHAPJJDNCTRGQGNTSELDGISMBOAICRGQOEAJEQBJOIHIKOCIHBIOOK
EOQQLEPCEANPRTHIATOJCEKTJ
MNDSEGLDRRJFKKEQSLHRQTHARREAGKDMPSDTHEKKITNQJLSHPNANJIQBIQLPBEAKESMIRNDDMSI
SPLERDGPGTFPIHCCNPIRGMKOK
TQCJKITSKKSONLHGEOBHABPFSDTOPCCTRSQEHCCIHLBTKHEDEFRTBRMTLBFFBJDRHOILOPPDGPQ
CTTHTBDTBHIITCRDLNEFLQKHM
SREFPGTBSTHEFLQPGDQFSDGQJCNMJCALSQERNPLGHOAJSOHPKIHRFNAIPESODALCDJALOSDTQFH
OCHQALPNPRQDLINIMDJQCFIOM
MMHPMDSLFIMFMMSPTHHKQSPCIMMLAEBGEJEFKBIEFJCMCNTJLMDLGDQIFOSRANSKSIBLDKOJBDK
QJNTFFTLHQLKOBKLNQNLERSOI
TPGOQGJRNOSAAJHPEPKIFGRJJGHBLCTQNTHTDJRDKBRMBGIRHGHONSCPHIODGBNPLTOESRJCLBT
HRRRDRRFFFHCBSMJMHGTNLPBJ
JFOKBGILNPKHCHMPBJLBGBDSGSKGGKOLHCSHMGBCMPAPBFJBIPPJSEQDSFNSIRFMJMEJETBCRCK

OMKHEHKFTQFNJBCACJDMJAOAE
JBAFTPELJIOCHJGOKLBGFGLANBBPOQDJDNPCPHGIJQHRJNNGFDERTEPRDGMELIAOJPNJRSFMQIJ
LMNOTEEOODBTOAHDCRTLNQGKO
BPQDRAMOTKOFBPMNFADFMIQISABEAOONTLIJSPRFJEDIKAFKQOMEHADKAANFRAKSKRMKRGHHPKD
AMGSNNDNBEMOTJSTSCAFBHGIK
HFCGRJKFHBGONCOKRLPBBIIRFGAOGRMOADCRHELDPOJBKIAGAFBTGSPRMJOJDLRQTEAAAQKLQFH
KAAFRSEBFTNIKERHSOHOFGFDT
GMPPJFFGBHATTCJLTRIQNHDOGSDKHKISHNTMHDKLHQKMPLEFLNGLCLHLRIFJLGKJATFDKDKGAEN
IFIHITOSLJASOBQLISSBRRJJK
KPCICOQLDRASPLEAELIATDBRQNRQJHEDTCJMFDQBTRCEGNNKMHCMSIECMELLGCBJQKTNDDPKSBJ
TNMAJJRHFEQPCMHGPGGQLHEFN
SJLKTAOJFDNJLEDFTSSAKJGOSLSTCPKDQSAFILMMMSNPJJCPJTSOLOMKGTFMSEKKHFNMDNDKKLH
JMQBTQDSHKMDJGFQKCGGMAHCF
NPMNJPIAFAAEENCTFEPFNLMNQGIEGDPGSBSPLBSNGNRMTGKKCRNNERHJOPHCDKJNGSBKOSTLTBG
SHNTIHJAJENMSJOMCNATAFKGN
MBSBCBSTCDQJGQQQKADMJTNQNMFSGDHENMARBAJBKTOGIDREJMFQGBJRLGTGLGHLHLMGRMEFEKD
FASMQAJNMINQPTEKGOCHGJTKT
OJSLHQFLIMMESDAJOOPDFTBJLNBJGEHSEMAQJAQJOFKQERAMSJJNIORDMHQQTFCKLNQKMOSBPHF
JODDPKBDSBSJMIRQLMTPDGTQS
JLJJTLNBQGNQKPLIQGFEKPCDJNSNHTENMHRPOEKBCOMJMDPFRMNTTMJKNMABQEKKNNIFDIJOHKK
QTMCMEJSJHECBJAANKHEAPGAK
LHJKFTHISSFAEMLPKFPBOMHOMDSRQSIJPOKSCMRRGSPPGRFLPJSISITEIBKMOINMRJRLSOOGPEA
QLCCCMMHMBCCEMFSAGEOCFDFA
HEJBBIJLQIQRTDMPFDQMNKNDQTPTFFOMADDBLIRAAOPOBFHTACPOPIEEHBJLLOFNMLKCODEDGSN
PCATLIIFMGRGHRRCKFTFKAPBR
DHOQBEITRNNPTIGECCDIAMIAJDSDEEBEEAELSNMBASELFIQFKKTIBBPOLPAMQNQAJABQFFLPFTE
PAQNGHHDNIBKGJTIJHTBRQEST
GTPAJQOTPREJFNOQEQFFOGFCDFQQHTRBAMLMILAHNPSFQFIBTFQARNEDOQKTEKIABDMLGLQLHBG
SMPSJTMINPTTAFEKLFATJLRJR
CKKHJAQMACDKJTMOARFBQMAJDDMFLKSGRARPHDIHBHHKBDMFQGTKGINAJTFKNRLKDHBIATLTDBD
PHHNEKPLERFMPRNGGNSJRQEOM
RIIJSBSLLFOSLMEBLTJARFNIKKHLMKNELJAHOOMRINPPMEODRBFGCCEHBBBFLSBBESBSKRSFHMA
CKHKIHITRIFOFKOQJILOFEFJH
ISDOINKFISDCJETRPJEJBSLMEDFGRLIACQPSLOADJBRTGQRGSDNTSNPLHCRTRPEPRMOIDQHMITJ
RPIGATCBPEOGIFBOEOSLKBBFM
SJCACFBLFBRDQLMNRJGFSAQMKHBTJMHCOJRMLLMBKLHLSOLRDPKRFGEJNIAALARDMEASTQLTGRK
ERORDPQPEHIFOBRSDNKSRMFIT
SAMJDMMLMMLIBHAJEBMHNLSTMFGBSIQHHMCLHENONTIOQJBOIOIMEKRIIKBDRCMHQMKFHMAOKIE
KKRDOHTFBDNSSIJAMGPBNDDOD
TMNAQCTJRAMFMLHDKRFJCRCKGCSDNOFBPBECDGSFOJDPJOPMBJCNITHHHQTPNAQEDTJOSEQQNFJ
EERAJHCCCPOMBNERLNOJOKRND
TPRQFEINOEOLRMIQDTEOSPDKLADISAAIETJHQFTESOSMMOJIQBTRQQRIMJKTNLAPKGBQANGKKIH
GSILQRJGNAEEQACPBTDLPFFNH

GHCTEKFPHPMJLHPIOLSQDCJIGPAERDLEOISICDJJNTEMTKBJCQGLFFIPFAEKFKBCQLODSTGOCTF
PCLJEAHNNBNJNDGCBHGGHHJHG

2、优化的 KMP 程序（python）

```
1、def kmp_pattern(seq):
2、    P = [0]*len(seq)
3、    j = 0
4、    for i in range(1, len(seq)):
5、        while j > 0 and seq[j] != seq[i]:
6、            j = P[j-1]
7、        if seq[j] == seq[i]:
8、            j += 1
9、        P[i] = j
10、        return P
11、
12、    def kmp_match(src, tmp, pat):
13、        ret = []
14、        j = 0
15、        for i in range(len(src)):
16、            while j > 0 and tmp[j] != src[i]:
17、                j = pat[j-1]
18、            if tmp[j] == src[i]:
19、                j += 1
20、            if j == len(tmp):
21、                ret.append(i-len(tmp)+1)
22、                j = pat[j-1]
23、        return ret
24、
25、    def kmp_mismatch_once(src, tmp, pat, i, j):
26、        p_i, p_j = i, j
27、        state = 0
28、        s_len, t_len = len(src), len(tmp)
29、        while i<len(src):
30、            if j<t_len and state == 0:
31、                if src[i] == tmp[j]:
32、                    j += 1
33、                else:
34、                    state = 1
35、                    p_i, p_j = i, j
36、                    j += 1
37、            elif j<t_len and state == 1:
38、                if src[i] == tmp[j]:
39、                    j += 1
40、                else:
41、                    #not match
42、                    i, j = p_i, p_j
43、                    while j>0 and tmp[j] != src[i]:
```

```
44、                    j = pat[j-1]
45、                return i, j, i-j, False
46、
47、            if j == len(tmp):
48、                ret = i - t_len + 1
49、                if state == 0:
50、                    j = pat[j-1]
51、                    return i+1, j, ret, True
52、                elif state == 1:
53、                    i, j = p_i, p_j
54、                    while j>0 and tmp[j] != src[i]:
55、                        j = pat[j-1]
56、                    return i, j, ret, True
57、            i += 1
58、        return i, j, 0, False
59、
60、
61、    def kmp_one_mismatch(src, tmp, pat):
62、        pre_i, pre_j, i, j = 0, 0, 0, 0
63、        pre_idx = -1
64、        idx, ret2 = 0, False
65、        ret = []
66、        while i<len(src):
67、            i, j, idx, ret2 =  kmp_mismatch_once(src, tmp, pat, i, j)
68、            if pre_i == i and pre_j == j and idx == pre_idx: #repeat
69、                i += 1
70、            else:
71、                pre_i, pre_j = i, j
72、                pre_idx = idx
73、                if ret2 == True:
74、                    ret.append(idx)
75、        return ret
76、
77、    sys.stdin=open( 'input00.txt', 'r')
78、    T = int(raw_input())
79、    while T:
80、        T -= 1
81、        src = raw_input()
82、        tmp = raw_input()
83、        if T != 0:
84、            emp = raw_input()
85、        src = list(src)
86、        tmp = list(tmp)
87、
88、        pat = kmp_pattern(tmp)
89、        idces = kmp_one_mismatch(src, tmp, pat)
```

```
90、        for v in idces:
91、            print(v),
92、        print('')
```