信息检索与数据挖掘

第4章 索引构建与索引压缩

——第二讲 索引压缩

课程内容

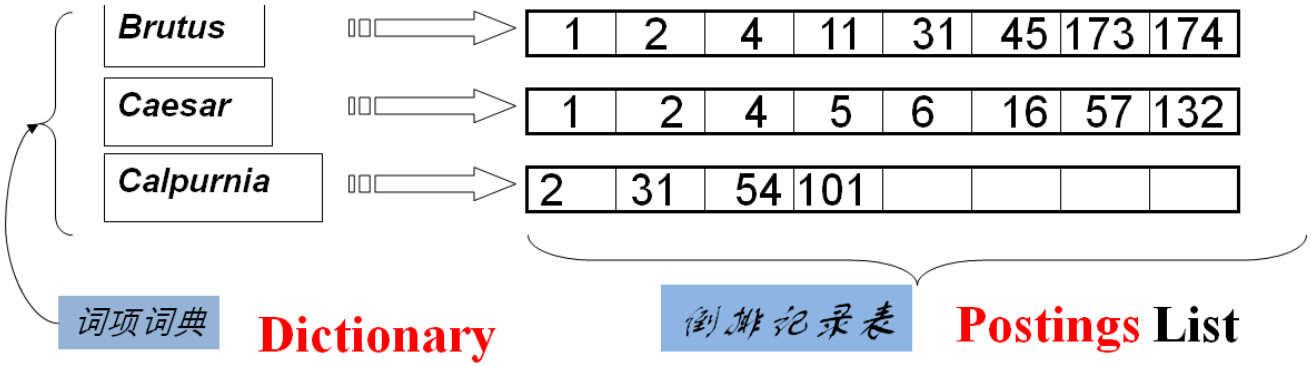
- 第1章　绪论
- 第2章 布尔检索及倒排索引
- 第3章 词典查找及扩展的倒排索引
- 第4章 索引构建和索引压缩
  - 索引构建
  - 索引压缩
- 第5章 向量模型及检索系统
- 第6章 检索的评价
- 第7章 相关反馈和查询扩展
- 第8章 概率模型
- 第9章 基于语言建模的检索模型
- 第10章 文本分类
- 第11章 文本聚类
- 第12章 Web搜索
- 第13章 多媒体信息检索
- 第14章 其他应用简介

索引压缩

- 统计信息(对RCV1语料库)
  - 词典和倒排记录表将会有多大?
- 词典压缩
- 倒排记录表压缩

为什么要压缩

怎么压缩



索引压缩

- **统计信息(对RCV1语料库)**
  - 词典和倒排记录表将会有多大?
  - Heaps定律：词项数目的估计
  - Zipf定律：对词项的分布建模
- **词典压缩**
  - 将词典看成单一字符串的压缩方法
  - 按块存储/前端编码
- **倒排记录表压缩**
  - 可变长字节码
  - 一元编码/ γ 编码

为什么要压缩(一般来说)?

- **节省磁盘空间**
  - 省钱
- **提高内存的利用率**
  - 提高速度
- **加快数据从磁盘到内存的传输速度**
  - [读取压缩数据][解压缩]比直接[读取未压缩的数据]快
  - 前提：解压缩算法要很快
    - 我们目前所用的解压缩算法在现代硬件上运行相当快

为什么要压缩倒排索引?

- **词典**
  - 压缩的足够小以便能够放入内存中
  - 当词典足够小时，我们也可以在内存中存储一部分的倒排记录表
- **倒排记录文件**
  - 减少所需的磁盘空间
  - 减少从磁盘读取倒排记录文件所需的时间
  - 大的搜索引擎在内存中存储了很大一部分的倒排记录表
    - 压缩可以让我们在内存中存储的更多
- **我们将设计各种基于IR系统的压缩架构**

回顾 Reuters-RCV1语料库

符号	含义	值
N	文档总数	800,000
L	每篇文档的平均词条数目	200
M	词项总数	400,000
	每个词条的平均字节数 (含空格和标点符号)	6
	每个词条的平均字节数 (不含空格和标点符号)	4.5
	每个词项的平均字节数	7.5
	倒排记录总数	160,000,000

索引参数 vs. 索引内容

	不同词项			无位置信息倒排记录			词条		
	词典			无位置信息索引			包含位置信息的索引		
	数目(K)	Δ%	T%	数目(K)	Δ%	T%	数目(K)	Δ%	T%
未过滤	484,494			109,971			197,879		
无数字	474,723	-2	-2	100,680	-8	-8	**179,158.2**	-9	-9
大小写转换	391,523	-17	-19	96,969	-3	-12	**179,157.8**	**0**	-9
30个停用词	391,493	**-0**	-19	83,390	-14	-24	121,858	-31	-38
150个停用词	391,373	**-0**	-19	67,002	-30	-39	94,517	-47	-52
词干还原	322,383	-17	-33	63,812	-4	-42	94,517	**0**	-52

讨论：0的原因？

无损 vs. 有损压缩

- 无损压缩：压缩之后所有原始信息都被保留。
  - 在IR系统中常采用无损压缩
- 有损压缩：丢掉一些信息
- 一些预处理步骤可以看成是有损压缩：大小写转换，停用词剔除，词干还原，数字去除。
- 第7章：那些削减的倒排记录项都不太可能在查询结果的前k个列表中出现。
  - 对于前k个返回结果来说，这几乎是无损的

有损还是无损与需求相关！！

词汇量 vs. 文档集大小

- **词项的词汇量有多大？**
  - 也就是说，有多少个不同的词？
- **我们可以假定一个上界吗？**
  - 实际上并不可以：长度为20的不同单词至少有$70^{20}=10^{37}$个
- **实际中，词汇量会随着文档集大小的增大而增长**
  - 尤其当采用Unicode编码时

词汇量 vs. 文档集大小

- Heaps定律：$M = kT^b$
- M是词项的数目，T是文档集中词条的个数
- 参数k和b的典型取值为：$30 \leqslant k \leqslant 100$和$b \approx 0.5$
- 词汇量大小M和文档集大小T在对数空间中，存在着斜率为½的线性关系
  - 在对数空间中，这是这两者之间存在的最简单的关系
  - 这是一个经验发现（“empirical law”）

Heaps定律是Heaps在1978年一本关于信息挖掘的专著中提出的。事实上，他观察到在语言系统中，不同单词的数目与文本篇幅（所有出现的单词累积数目）之间存在幂函数的关系，其幂指数小于1。

Heaps定律

M：词项总数
T：词条总数

对RCV1文档集来说，虚线 $\log_{10}M = 0.49\log_{10}T + 1.64$ 是基于最小二乘法的最佳拟合结果。

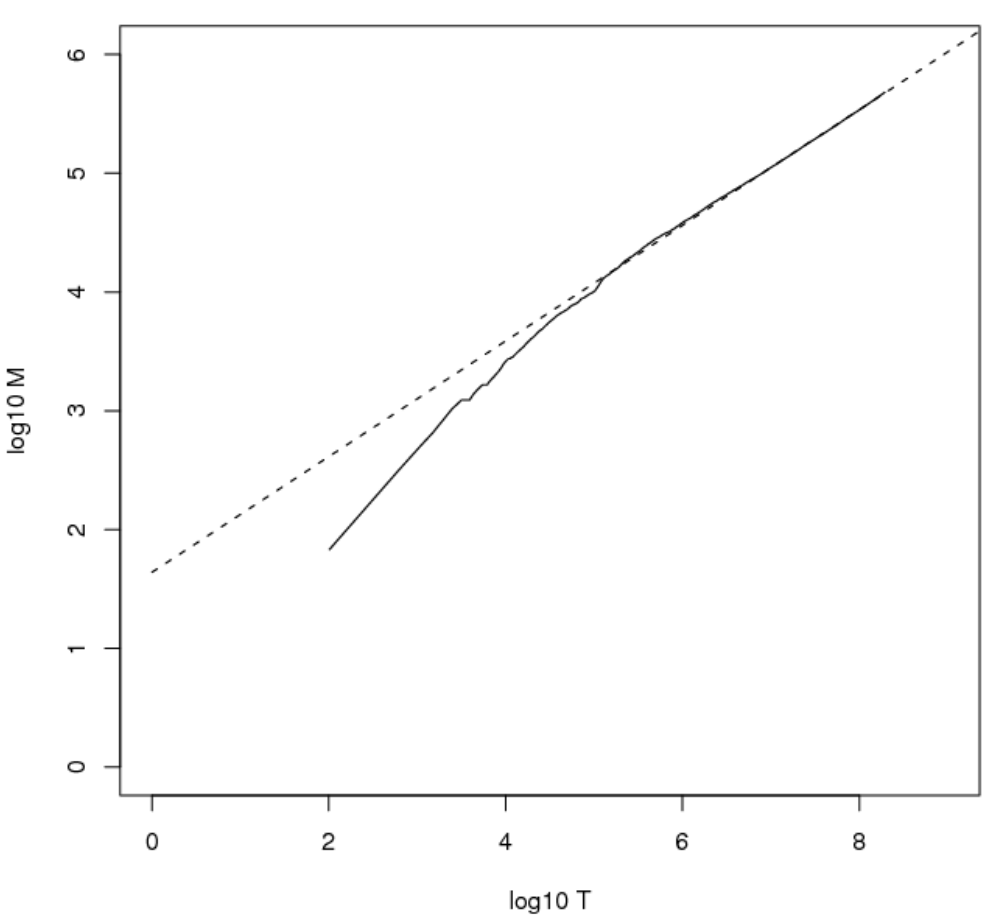
则 $M = 10^{1.64}T^{0.49}$，所以 $k = 10^{1.64} \approx 44$, $b = 0.49$

对RCV1是一个很好的经验拟合！

对于前1,000,020个词条，

Heaps定律会估计得到大约38,323个词项；

而实际数目是38365，和估计值非常接近



$$\log_{10}M = 0.49 \times \log_{10}T + 1.64$$

Zipf定律

- Heaps定律提供了对文档集中词汇量的估计
- 我们还想了解词项在文档中的分布情况
- 在自然语言中，只有很少一些非常高频的词项，而其它绝大部分都是很生僻的词项。
- Zipf定律：排名第i多的词项的文档集频率与1/i成正比
- $cf_i \propto 1/i = K/i$, K是一个归一化常数
- Cf_i是文档集频率：**词项**t_i在文档集中出现的次数

Zipf定律是Zipf在1949年的一本关于人类定位的最小作用原理的书中首先提出的，其中最令人难忘的例子是在人类语言中，如果以单词出现的频次将所有单词排序，用横坐标表示序号，纵坐标表示对应的频次，可以得到一条幂函数曲线。这个定律被发现适用于大量复杂系统。

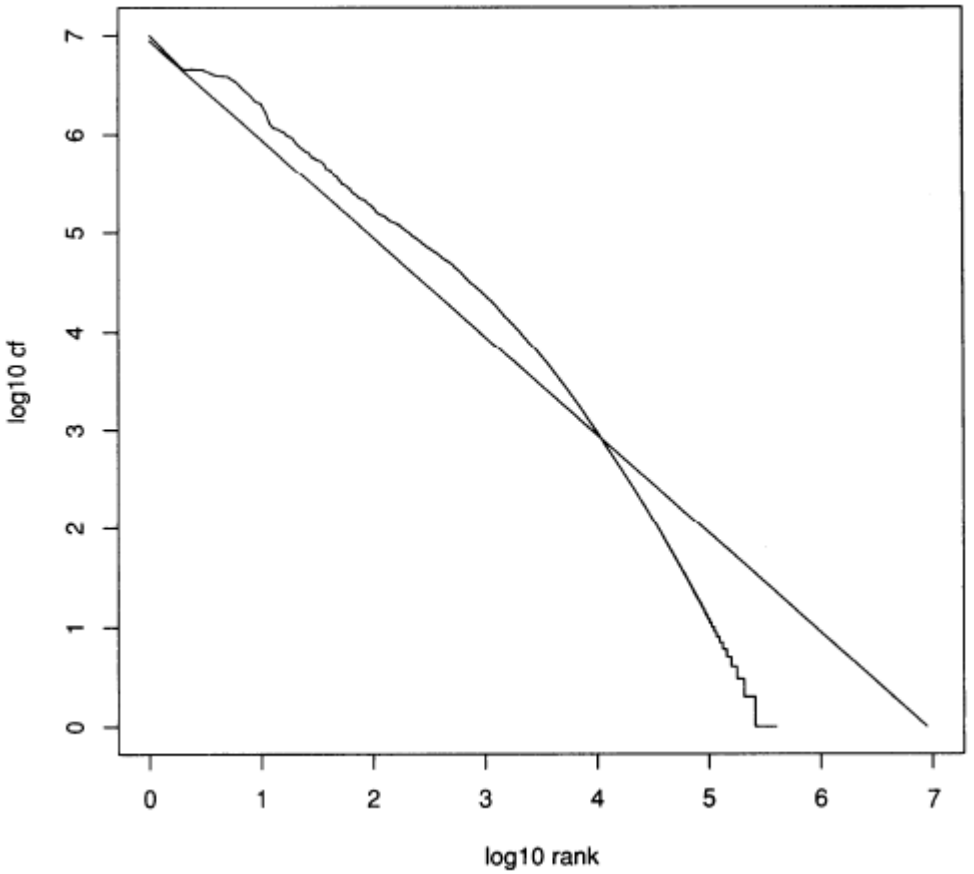
Zipf定律推论

- 如果最高频的词项(the)出现了cf_1次
  - 那么第二高频的词项(of)出现了$cf_1/2$次
  - 第三高频的词项(and)出现了$cf_1/3$次
- 等价的：$cf_i = K/i$ 中K是归一化因子，所以
  - $\text{Log } cf_i = \log K - \log i$
  - $\log cf_i$和$\log i$之间存在着线性关系
- 另一个幂定律关系

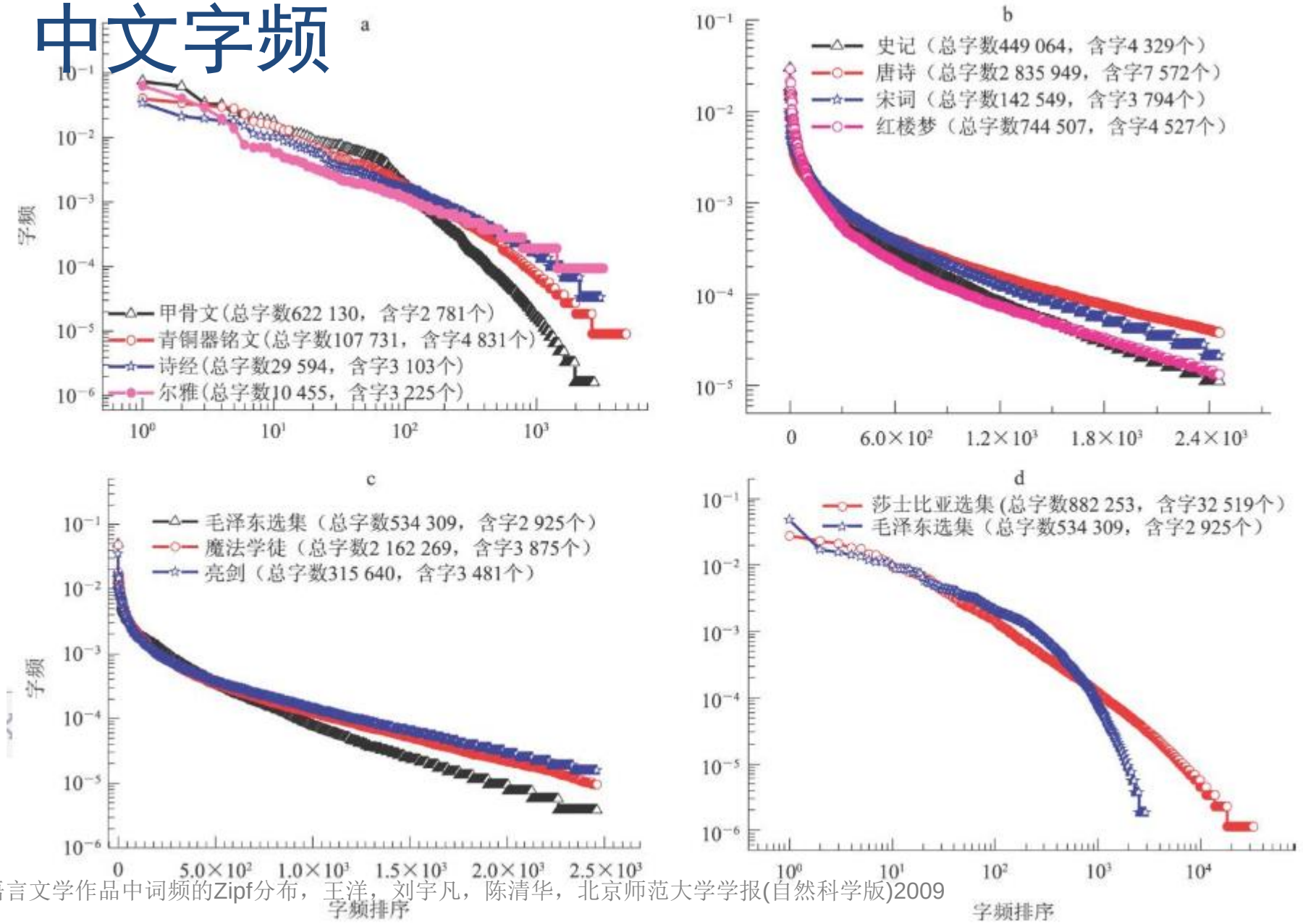
Reuters-RCV1文档集上的Zipf定律

cf：词项 t_i在文档集中出现的次数

rank：词项的排名



中文字频



汉语言文学作品中词频的Zipf分布，王洋，刘宇凡，陈清华，北京师范大学学报(自然科学版)2009

中文词频规律示例

表 1 文学作品中的词频

排序	《红楼梦》		《毛泽东选集》		《邓小平文选》		《莎士比亚全集》	
	词	词频	词	词频	词	词频	词	词频
1	了	0 040 73	的	0 085 48	的	0 079 46	the	0 031 83
2	的	0 027 15	和	0 016 42	了	0 015 70	and	0 029 37
3	我	0 010 88	在	0 015 04	我们	0 015 12	I	0 023 54
4	不	0 010 66	了	0 014 50	是	0 012 66	to	0 021 55
5	贾	0 010 10	是	0 011 57	在	0 011 49	of	0 019 51
6	道	0 008 69	一	0 005 82	和	0 010 40	a	0 016 56
7	你	0 008 50	我们	0 005 78	要	0 008 55	you	0 015 69
8	也	0 007 81	中国	0 005 44	问题	0 007 79	my	0 013 92
9	宝玉	0 007 75	为	0 004 88	有	0 007 43	in	0 012 27
10	来	0 007 67	有	0 004 48	不	0 007 29	that	0 012 17
11	又	0 007 11	不	0 004 39	党	0 006 43	is	0 010 35
12	便	0 006 65	地	0 004 17	就	0 005 45	not	0 009 59
13	说	0 006 30	革命	0 004 08	工作	0 005 22	me	0 008 78
14	这	0 006 09	而	0 003 92	也	0 005 05	it	0 008 75
15	去	0 005 55	国民党	0 003 91	一个	0 004 36	with	0 008 70

汉语言文学作品中词频的Zipf分布，王洋，刘宇凡，陈清华，北京师范大学学报(自然科学版)2009

题外话

- 很多复杂系统同时满足Zipf定律和Heaps定律，但是对于两者关系，学术界存在长期争论。通过一些随机过程模型，有些学者认为Zipf定律是本质的，Heaps定律是衍生的，可以从Zipf定律推出；有些学者（Zanette, Moutemurro）认为Heaps定律是本质的，Zipf定律是衍生的；有的学者认为这两种定律相互独立。
- 我们不依赖于任何随机过程，证明了Zipf定律更本质，而Heaps定律是衍生律。进一步地，我们证明了以前的两个定律指数之间的解析关系，只是在Zipf指数远大于1或远小于1或系统规模无穷大的时候的一种渐进解。遗憾的是，真实系统不满足三种条件中的任何一种。我们提出了新的解析方法，得到了更精确的解析结果，在35个真实数据中进行验证，发现有34个数据新结果都好于以前的结果。
- Linyuan Lü, Zi-Ke Zhang, Tao Zhou, “Zipf’s Law Leads to Heaps’ Law: Analyzing Their Relation in Finite-Size Systems”, PLoS ONE 5 (2010) e14139.

关于数字的统计规律

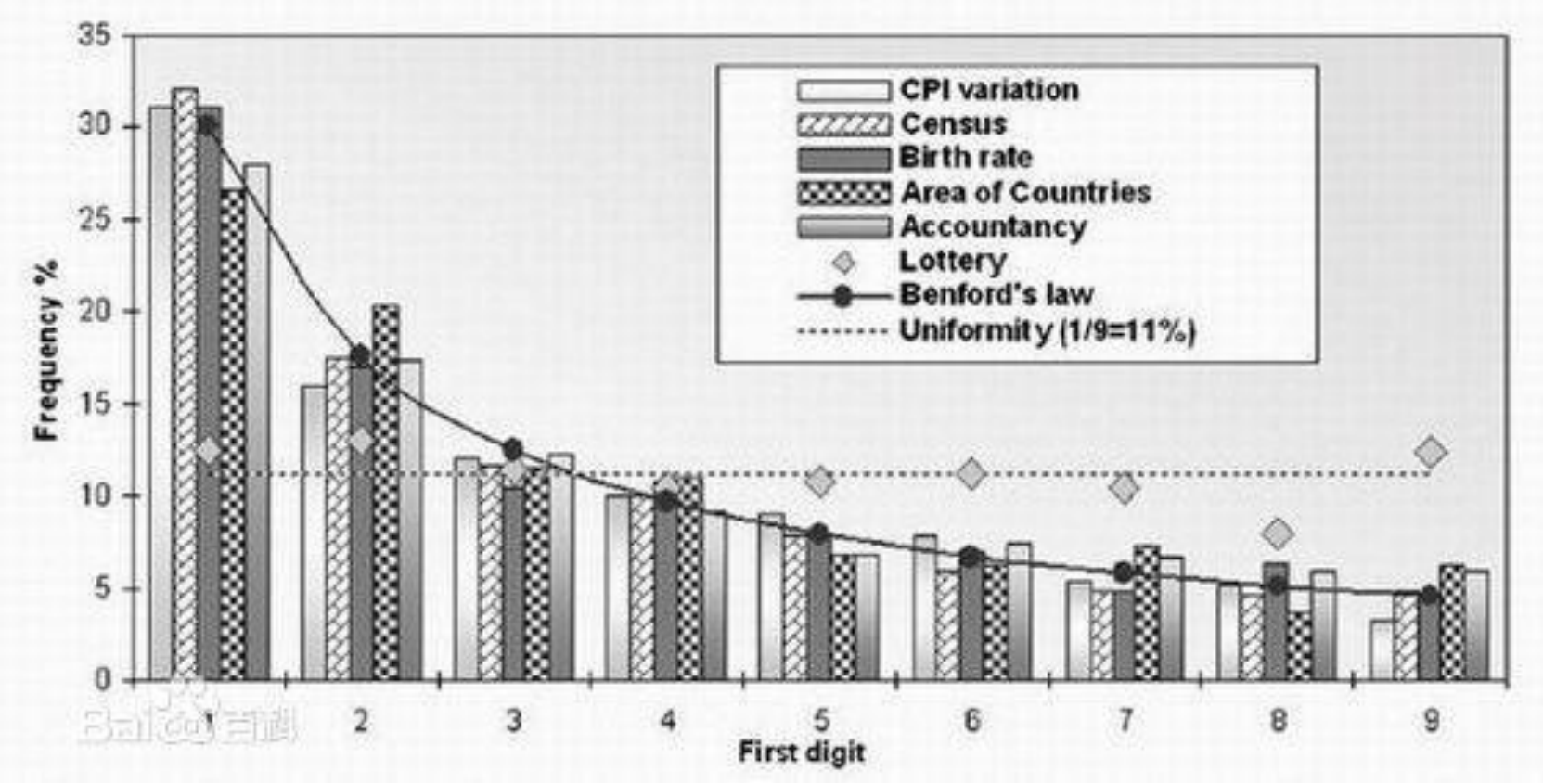
第一数字定律（Benford law）

描述的是自然数1到9的使用频率F(d)=log[1+(1/d)]（d为自然数），其中1使用最多接近三分之一，2为17.6%，3为12.5%，依次递减，9的频率是4.6%。

科学家们仔细研究第一数字定律后，无法对这种现象做出合理解释。定律的主要奠基人Frank Benford对人口出生率、死亡率、物理和化学常数、素数数字等各种现象进行统计分析后发现，由度量单位制获得的数据都符合第一数字定律。当然彩票上随机数据并不符合。

d	p
1	30.1%
2	17.6%
3	12.5%
4	9.7%
5	7.9%
6	6.7%
7	5.8%
8	5.1%
9	4.6%

Benford law



Benford law还在会计作假帐的审查和政治选票合法性审查起到了重要作用。

小结：语言统计学三大定律

Zipf law，Heaps law和Benford law

Zipf law：在给定的语料中，任意一个term，其频度(freq)的排名（rank）和freq的乘积大致是一个常数。

Heaps law：在给定的语料中，独立的term数(vocabulary的size)v(n)大致是语料大小(n)的一个指数函数。

Benford law：在自然形成的十进制数据中，任何一个数据的第一个数字d出现的概率大致$\log_{10}(1+1/d)$

压缩

- **现在，我们考虑压缩词典和倒排记录表**
  - 仅仅考虑基本的布尔索引
  - 不研究包含位置信息的索引
  - 我们将考虑压缩架构

索引压缩

- 统计信息(对RCV1语料库)
  - 词典和倒排记录表将会有多大?
  - Heaps定律：词项数目的估计
  - Zipf定律：对词项的分布建模
- 词典压缩
  - 将词典看成单一字符串的压缩方法
  - 按块存储/前端编码
- 倒排记录表压缩
  - 可变长字节码
  - 一元编码/ γ 编码

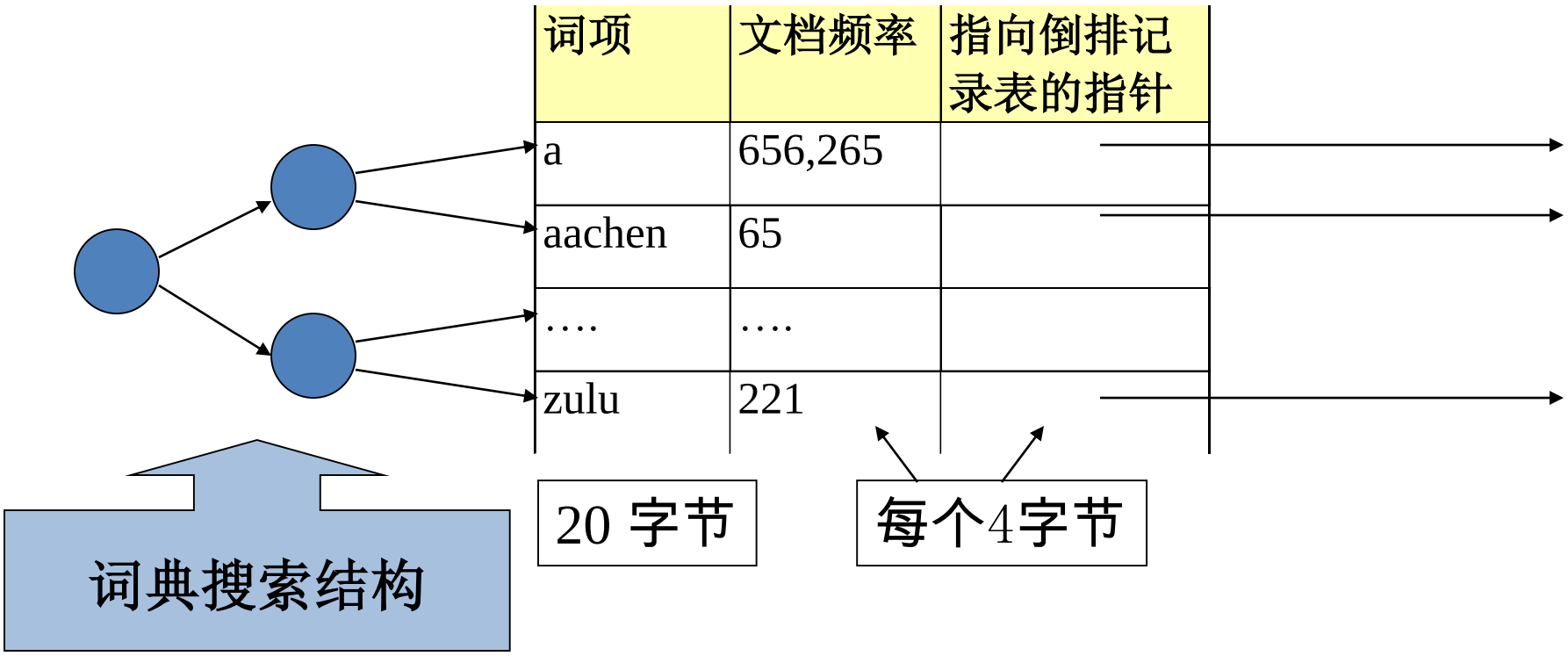
为什么要压缩词典？

- 搜索从词典开始
- 我们想将词典放入内存中
- 和其他应用程序共享内存资源
- 手机或者嵌入式设备通常只有很小的内存
- 即使词典不存入内存中，我们也希望它能比较小，以便搜索能快速启动
- 所以，压缩词典非常重要

尽管即使是非常大规模的文档集的词典也往往能够放入一台标准台式计算机的内存，但是在很多其他场景下情况并非如此。例如，大公司的一台企业搜索服务器也许要索引数太字节的文档，由于文档中可能包含多种不同语言，所以最后的词汇量可能会很大。

词典存储

- 定长数组存储
  - 400,000词项；20字节/词项 = 11.2 MB



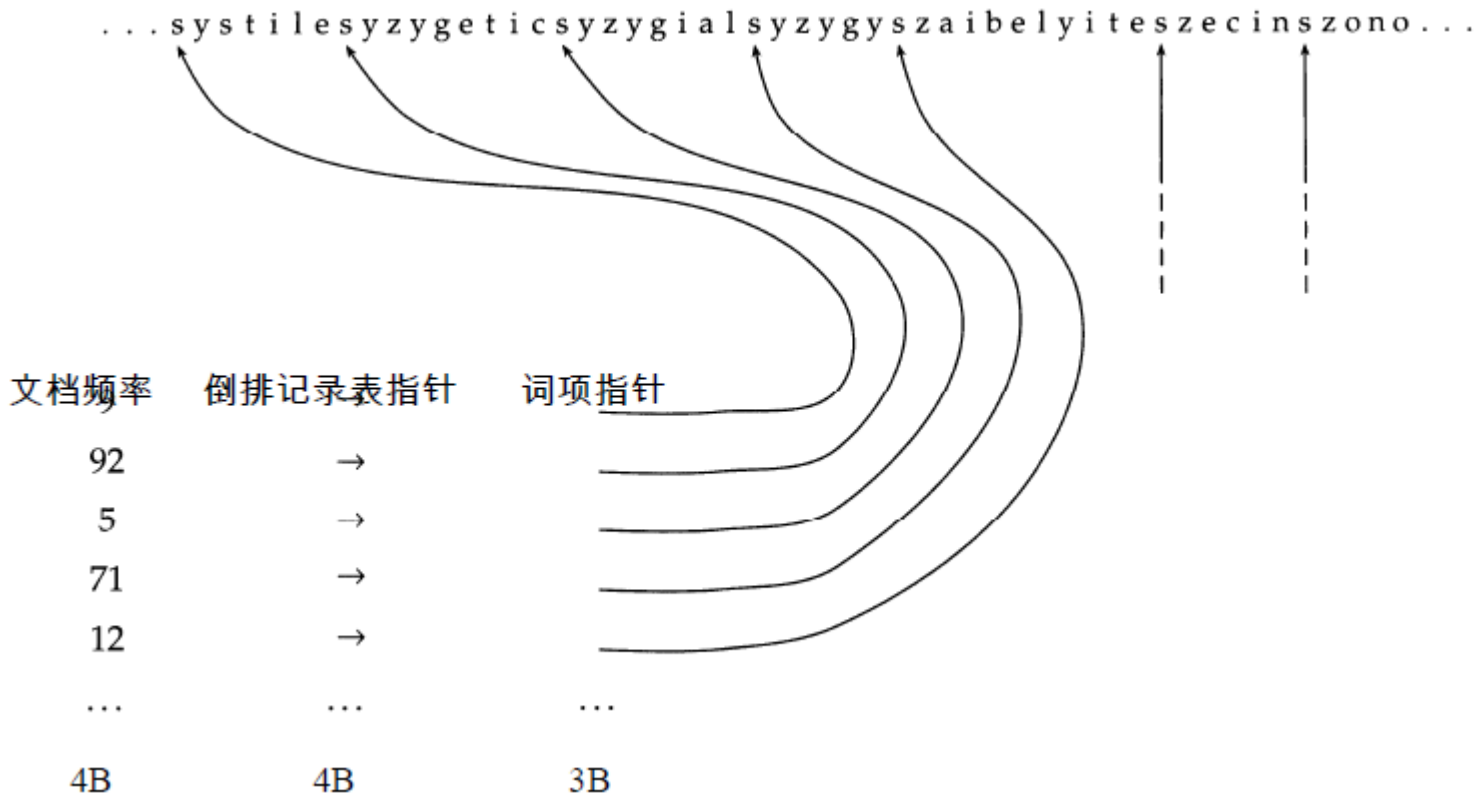
定长方法存储词项浪费空间

- 在词项那一列大部分的字节都被浪费 — 我们为每个词项分配了20字节的固定长度。
  - 但我们仍然不能解决“*supercalifragilisticexpialidocious*”和“*hydrochlorofluorocarbons*”
- 书面英文中单词的平均长度约为4.5个字符
  - 练习：为什么不用这个值来估计词典的大小？
- 英语中平均的词典词项长度为8个字符
  - 平均会有12个字符的空间浪费
- 较短的词项支配了词条的数目但是并不是典型的平均值

hydro-chlorofluorocarbons
氢氯氟烃
supercalifragilisticexpialidocious
奇妙的; 难以置信的

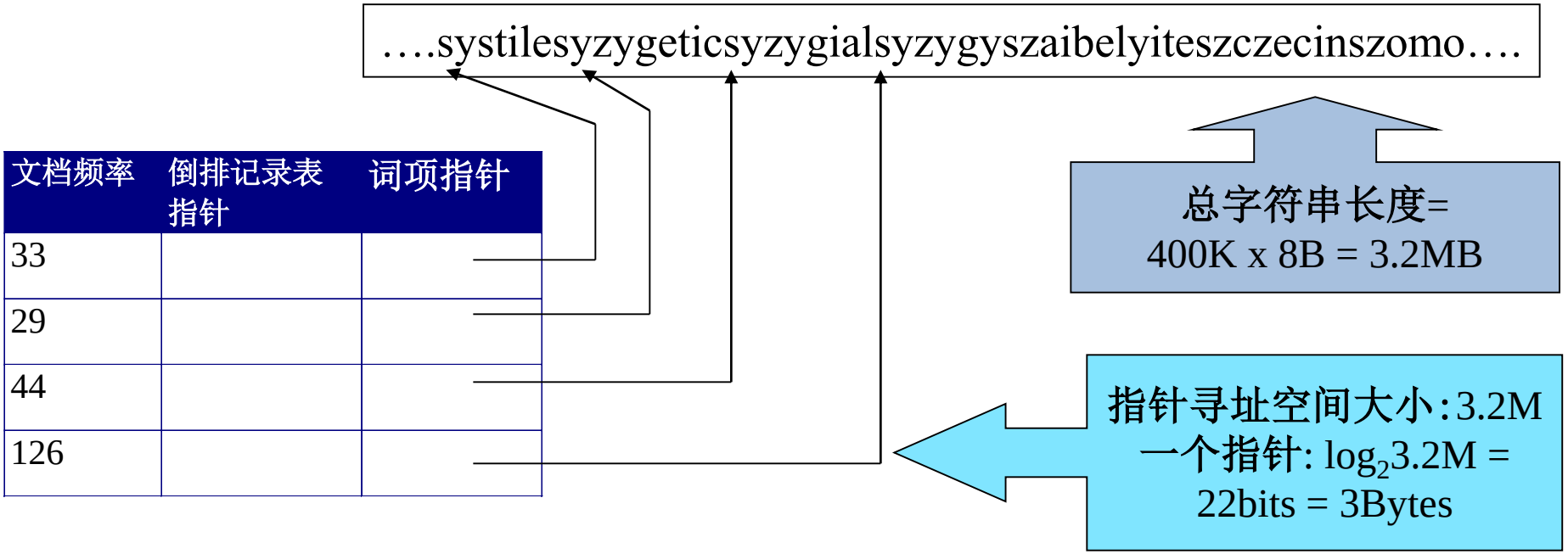
定长→不定长：定位指针的思路

一种解决上述缺陷的方法是，将所有的词项存成一个长字符串并给每个词项增加一个定位指针，它在指向下一词项的指针同时也标识着当前词项的结束。



压缩词项列表：将词典看成单一字符串（Dictionary-as-a-String）

- **将词典存储为一个长字符串：**
  - 指向下一词项的指针同时也标识着当前词项的结束
  - 期望节省60%的词典空间



字符串词典的空间大小

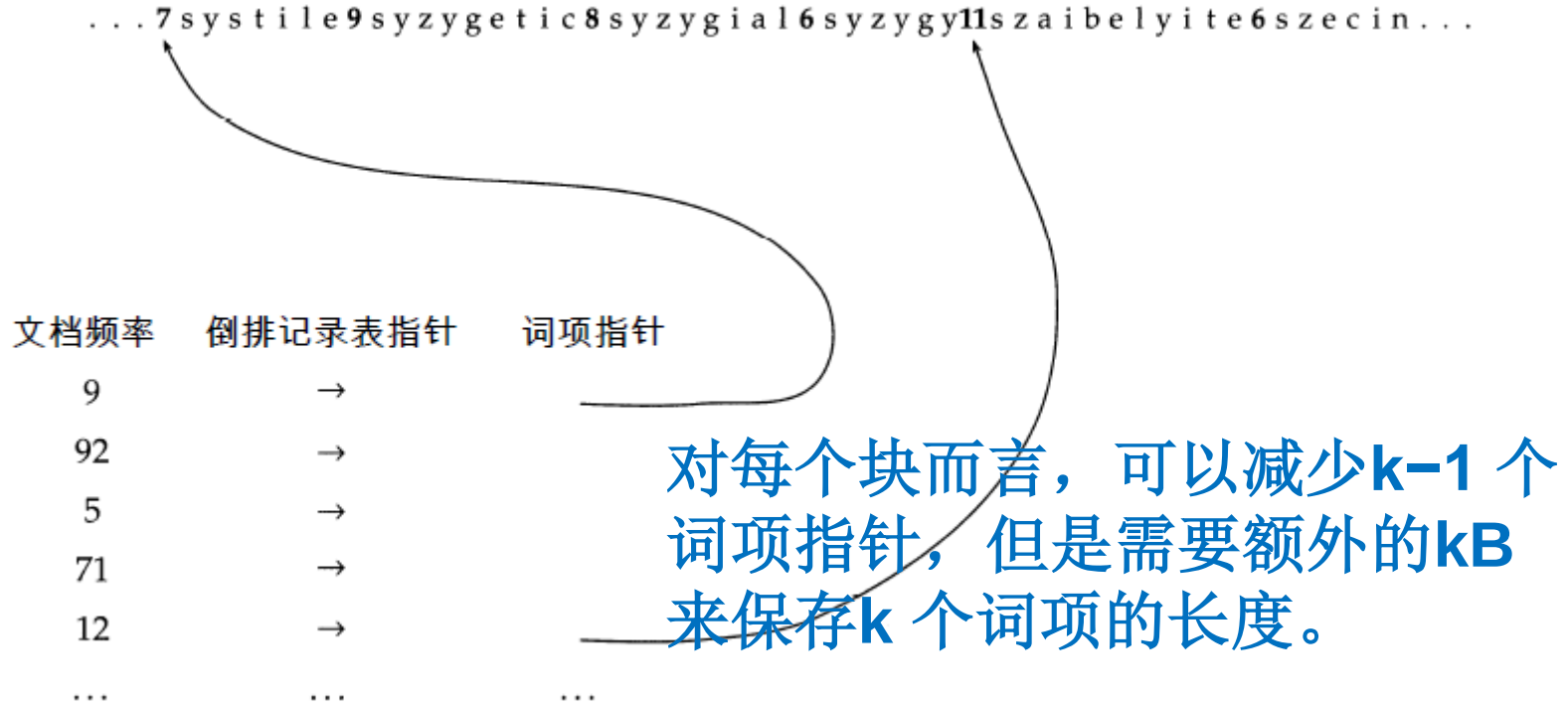
- 词项文档频率 – 4字节
- 倒排记录表指针 – 4字节
- 词项指针 – 3字节

现在是每个词项
平均占用11字节
而不是20字节

- 在词项字符串中，每个词项平均8个字节
- 400k词项 $\times 19 = 7.6MB$(固定长度时为$11.2MB$)

定位指针的长度能够减小？

进一步的压缩：将长字符串中的词项进行分组变成大小为k 的块（即k 个词项一组），**然后对每个块只保留第一个词项的指针**。同时，我们用一个额外字节将每个词项的长度存储在每个词项的首部。



按块存储（Blocking）

- 我们为每第k个词项字符串存储一个指针
  - 下面的例子：k=4
- 需要存储词项长度(额外1字节)

文档频率	倒排记录表指针	词项指针
33		
29		
44		
126		
7		

节省3个指针的开销 – 9字节

损失了4字节存储词项长度.

节省空间

- 我们取块大小k = 4
- 不采用按块存储时，每个指针花费3字节
  - 3 × 4 = 12字节

  现在我们花费3 + 4 = 7字节

削减了另外的0.5MB空间。原来的7.6MB存储空间可以进一步降低到7.1MB

讨论：

K=8？

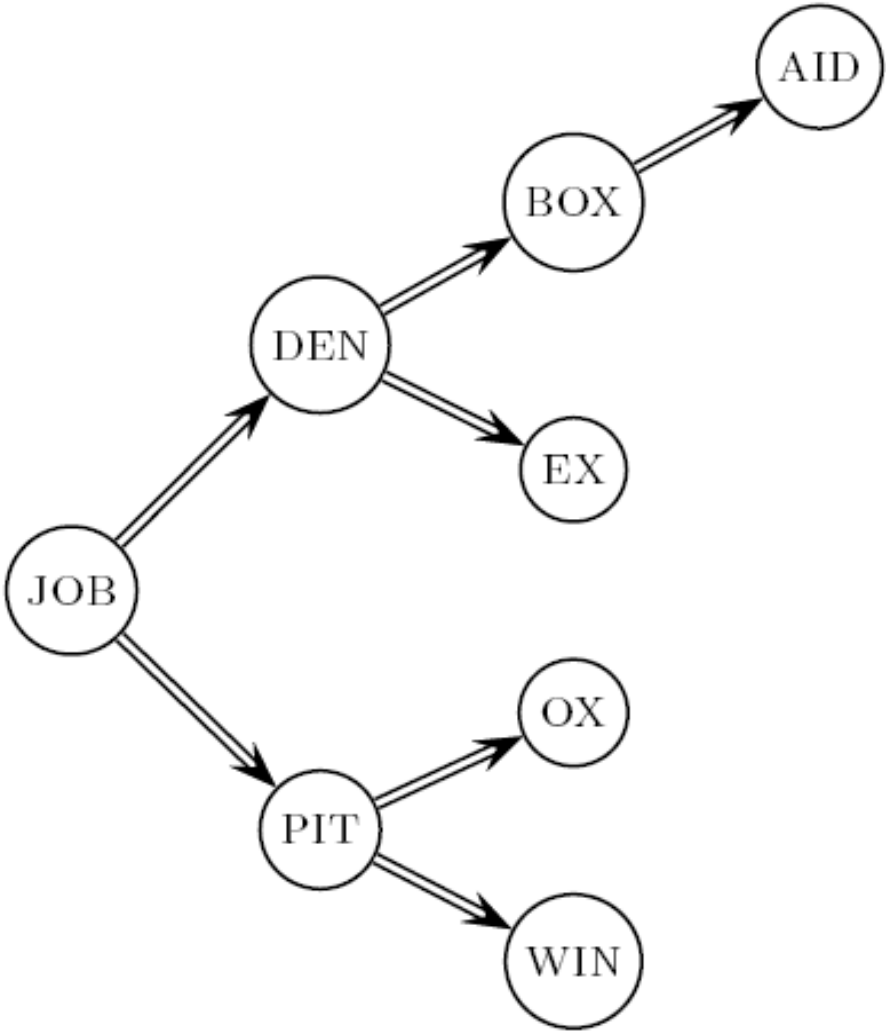
K=16？

为什么不选取更大的k?

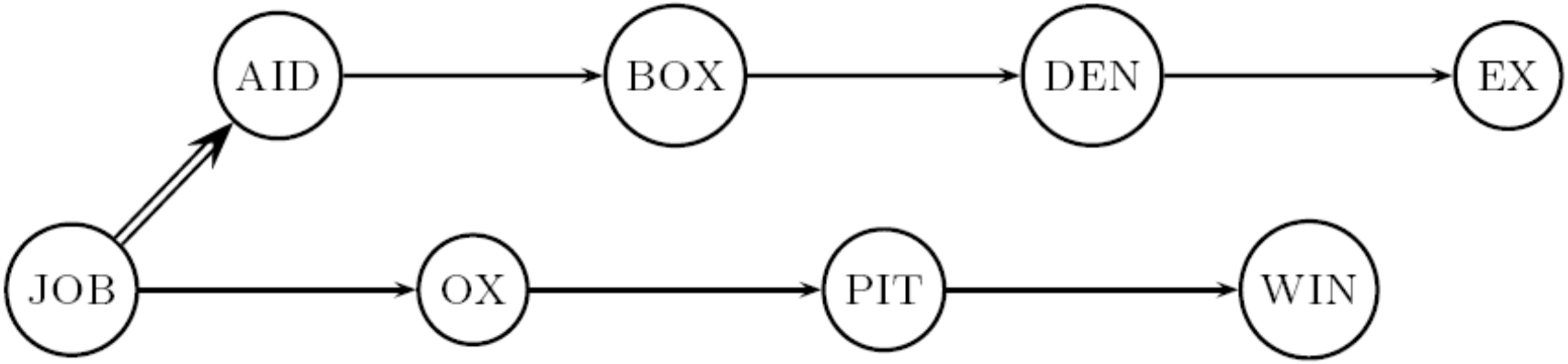
未压缩词典的搜索

- 假设词典中每个词项被查询的概率相同（实际中并非如此！），
- 平均比较次数=

(1+2*2+4*3+4)/8=2.6



按块存储方式下的词典搜索



- 二分查找只能在块外进行
  - 然后在块内进行线性查找得到最后的词项位置
- 块大小为4(二分树)，平均比较次数=

  (1+2*2+2*3+2*4+5)/8 = 3

前端编码（Front coding）

- **前端编码：**
  - 按照词典顺序排列的连续词项之间往往具有公共前缀。
  - （块内k个词项的最后k-1个）

8automata8automate9automatic10automation

→8*automat*a*1◊*e*2◊*ic*3◊*ion*

编码 ***automat***

除***automat***外的额外长度

类似一般的字符串压缩方法

小结词典压缩：RCV1文档集的词典压缩结果

数据结构	压缩后大小(MB)
定长数组	11.2
长字符串+词项指针	7.6
按块存储，k=4	7.1
按块存储+前端编码	5.9

索引压缩

- **统计信息(对RCV1语料库)**
  - 词典和倒排记录表将会有多大?
  - Heaps定律：词项数目的估计
  - Zipf定律：对词项的分布建模
- **词典压缩**
  - 将词典看成单一字符串的压缩方法
  - 按块存储/前端编码
- **倒排记录表压缩**
  - 可变长字节码
  - 一元编码/ γ 编码

倒排记录表压缩

- 倒排记录表远大于词典，至少为10倍
- 迫切要求：紧密地存储每一个倒排记录表
- 每个倒排记录用文档ID来定义
- 对Reuters (800,000文档)来说，当使用4字节整数表示时，每个文档ID需要32bit
- 或者，我们可以用$\log_2 800{,}000 \approx 20$ bits 来表示每个文档ID
- 我们的目标：用远小于20bit来表示每个文档ID

Reuters-RCV1：800 000 篇文档，倒排记录数目为100 000 000

100M*20/8=250MB

预处理前后词项、词条数目

	不同词项			无位置信息倒排记录			词条		
	词典			无位置信息索引			包含位置信息的索引		
	数目(K)	Δ%	T%	数目(K)	Δ%	T%	数目(K)	Δ%	T%
未过滤	484			109,971			197,879		
无数字	474	-2	-2	100,680	-8	-8	**179,158.2**	-9	-9
大小写转换	392	-17	-19	96,969	-3	-12	**179,157.8**	**0**	-9
30个停用词	391	**-0**	-19	83,390	-14	-24	121,858	-31	-38
150个停用词	391	**-0**	-19	67,002	-30	-39	94,517	-47	-52
词干还原	322	-17	-33	63,812	-4	-42	94,517	**0**	-52

倒排记录表：相反的两点

- 像“*arachnocentric*”这样的词项可能在一百万个文档中才会出现一次 - 我们可以用

  $\log_2 800,000 \approx 20$ bits来存储这一倒排记录中的文档ID。
- 像“the”这样的词项在每个文档中都会出现，所以对它采用20bit存储文档ID太浪费了
  - 这种情况更希望是0/1的bit向量

规律的探寻：倒排记录表项中文档ID的间距（GAP）

- 我们按照文档ID的递增顺序来存储一个词项的倒排列表。
  - *Computer*：33, 47, 154, 159, 202…
- 结论：可以存储间距
  - 33, 14, 107, 5, 43…
- 期望：绝大多数间距存储空间都远小于20bit

找找GAP：3个倒排记录表项

	encoding	postings list					
THE	docIDs	...	283042	283043	283044	283045	...
	gaps			1	1	1	...
COMPUTER	docIDs	...	283047	283154	283159	283202	...
	gaps			107	5	43	...
ARACHNOCENTRIC	docIDs	252000	500100				
	gaps	252000	248100				

可变长度编码

- 目标：
  - 对于*arachnocentric*，我们使用20bit/间距项
  - 对于*the*，我们使用1bit/间距项
- 如果一个词项的平均间距为G，我们想使用 $\log_2 G$ bit/间距项
- 关键问题：需要利用整数个字节来对每个间距编码
- 这需要一个可变长度编码
- 可变长度编码对一些小数字使用短码来实现这一点

GAP→可变字节码(Variable Byte)

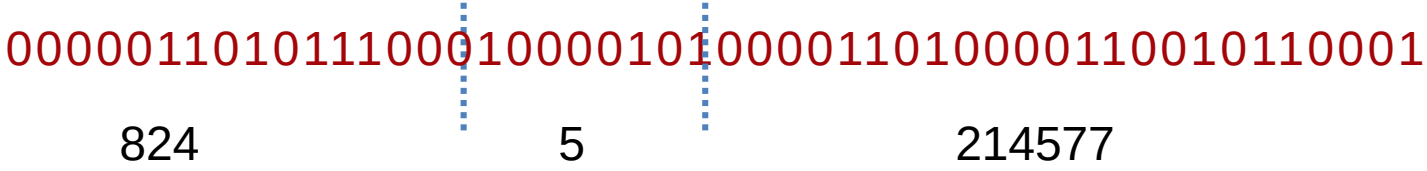
- 对一个间距值G，我们想用最少的所需字节来表示 $\log_2 G$ bit
- 每个字节（8bits）的最高比特作为延续位c，c=0意味着编码未结束，后面还有字节，c=1结束
  - 若$G \leqslant 127$,对7位有效码采用二进制编码并设延续位c=1
  - 若G>127,则先对G低阶的7位编码，然后采取相同的算法用额外的字节对高阶bit位进行编码
- 设置最后一个字节的延续位为1(c=1)，其他字节的c=0

何谓可变字节编码？即编码后的结果的字节数可变，可以是1个字节或多个字节，需要有办法区分出到底是多少个字节

例子

文档ID	824	829	215406
间距		5	214577
VB 编码	00000110 10111000	10000101	00001101 00001100 10110001

倒排索引以一连串字节的形式存储



关键特性：VB编码过的倒排记录表是唯一前缀可解的

对一个小的间距(5)，VB编码使用了一整个字节，浪费存储空间

其它的可变单位编码

- VB编码思想也可应用在与字节不同的单位上：32bit(words)，16bit，4bit(nibble)
- 可变字节编码在那些很小的间距上浪费了空间 —半字节在这种情况下表现得更好
- 可变字节编码：
  - 被很多商业/研究系统所使用
  - 实现简单，能够在时间和空间之间达到一个非常好的平衡点

一元编码(Unary code)

- 数n用n个1后面加个0表示
- 3的一元编码为1110
- 40的一元编码为：

  11111111111111111111111111111111111111110
- 80的一元编码为：

  11111111111111111111111111111111111111111111
  11111111111111111111111111111111111111110

- 这看起来很没前途，但是……

γ编码

- 采用bit级别的编码，我们可以压缩的更好
  - γ编码是这些编码方法当中最好的
- 将间距G表示成长度和偏移的组合
- G的偏移是G的二进制编码，但是前端的1被去掉
  - 例如：13 -> 1101 -> 101
- G的长度是指偏移的长度
  - 13的偏移为101，所以长度即为3
- 长度部分的一元编码是1110
- 13的γ编码是长度和偏移的连接：1110101

γ编码例子

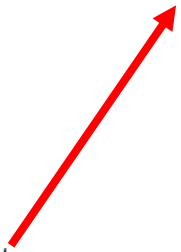
数字	长度	偏移	γ编码
0			none
1	0		0
2	10	0	10,0
3	10	1	10,1
4	110	00	110,00
9	1110	001	1110,001
13	1110	101	1110,101
24	11110	1000	11110,1000
511	111111110	11111111	111111110,11111111
1025	11111111110	0000000001	11111111110,0000000001

γ编码 – 解码

- 首先读入一元编码直至遇到0结束
  - 如对1110101解码时，一开始读入4位1110
- 通过一元编码计算后面偏移部分的长度
  - 1110计算出偏移部分长度为3
- 读入偏移部分，补上前端的1得到二进制编码
  - 读入101->1101=13

γ编码特性

universal code



- G的编码长度为2 ⌊log G⌋ + 1 bit
  - 偏移部分的编码长度是⌊log G⌋ bit
  - 长度部分的编码长度是⌊log G⌋ + 1 bits
- γ编码的长度永远都是奇数位
- 它与前面提到的最优化编码长度$\log_2 G$只差一个因子2
- 和VB编码一样，γ编码也是前缀唯一可解的
- γ编码对任何分布都适用
- γ编码是没有参数的

γ编码很少在实际中使用

- γ编码技术能够取得比可变字节更高的压缩率，但是解压的消耗会更高
  - 机器一般都有字边界 - 8, 16, 32, 64 bit
  - 超过字边界的操作都很慢
  - 在bit粒度下进行压缩和控制将会很慢
- 可变字节编码和γ编码相反，因此会更加高效
- 即使不考虑效率问题，可变字节码虽然有很小的额外空间开销，但是它在概念上更简单

RCV1压缩

数据结构	Size in MB
词典，定长数组	11.2
词典，长字符串+词项指针	7.6
词典，按块存储, k = 4	7.1
词典，按块存储+前端编码	5.9
文档集(文本、XML标签等)	3,600.0
文档集(文本)	960.0
词项关联矩阵	40,000.0
倒排记录表，未压缩(32位字)	400.0
倒排记录表，未压缩(20bit)	250.0
倒排记录表，可变字节码	116.0
倒排记录表，γ编码	101.0

总结

- 我们现在可以为布尔查询创建一个索引，既高效又非常节省空间
- 只有文档集总大小的4%
- 在文档集中只有文本总大小的10-15%
- 但是，我们忽略了索引的位置信息
- 因此，在实际中，索引所节省的空间并没有这么多

思考

- γ编码为什么是通用性编码？
- γ编码对倒排索引进行压缩能达到多高的压缩比？
- 习题5-5、习题5-8