

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

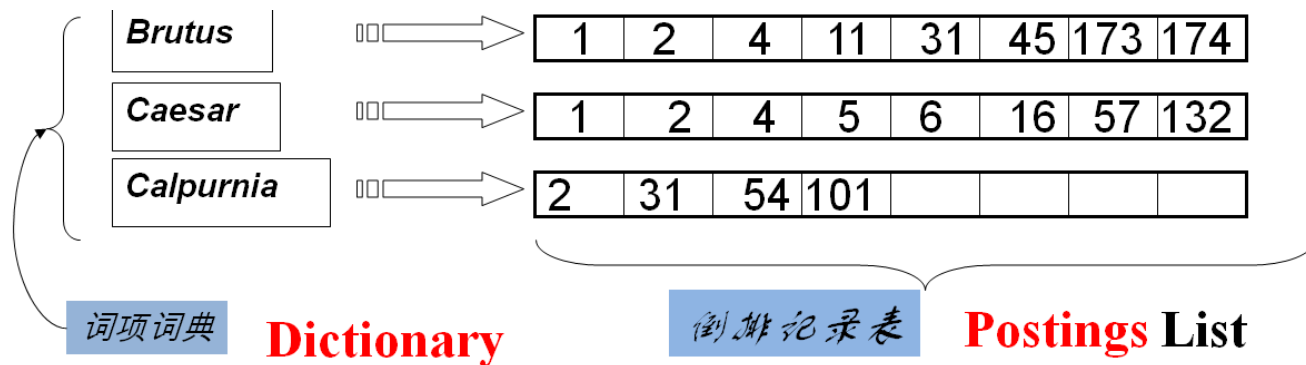
回顾4-1：语料库介绍 – 索引是硕大的

Reuters-RCV1语料的统计数据

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 (Terms) 总数	400,000
	每个词条 (Tokens) 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
T	词条 - 文档ID记录数 (即总词条数目)	800000*200=160 000 000

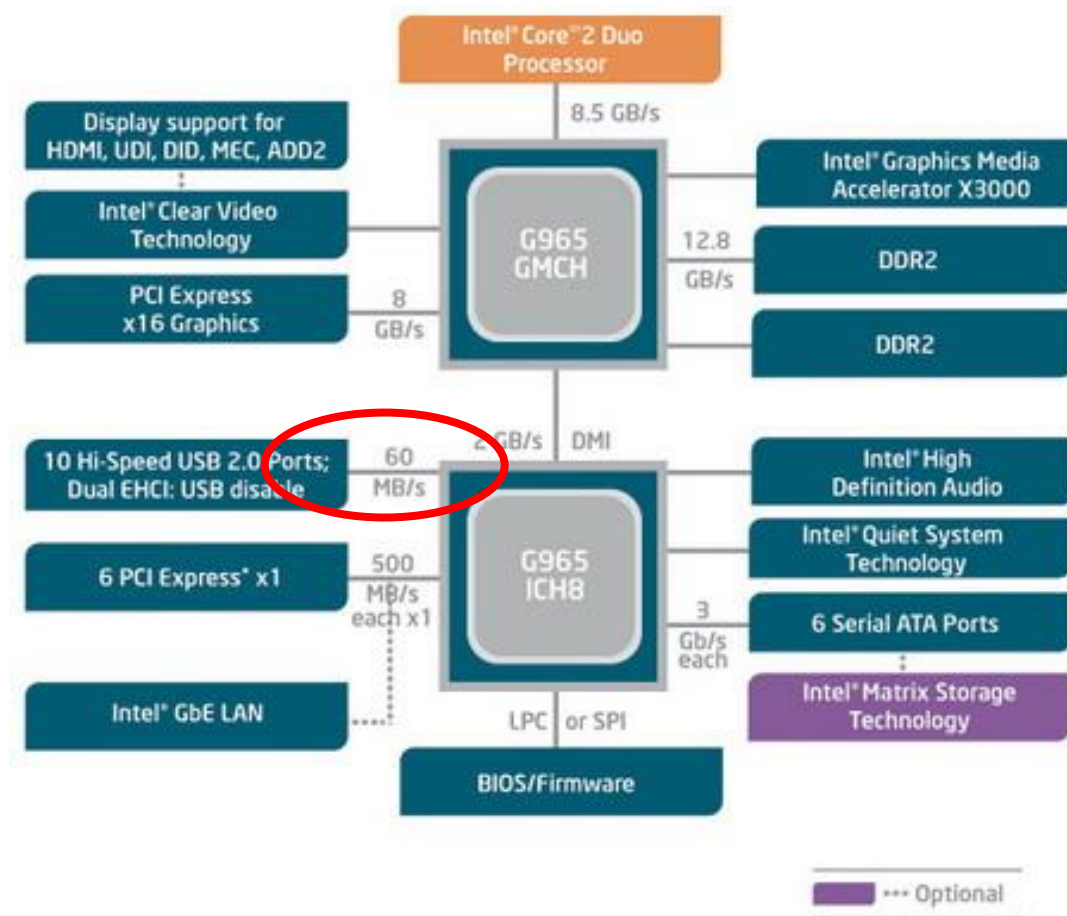
2^{19.6}2^{20+7.3}

1.28GB?



回顾4-2：硬件 – 并行设计思想很强大

计算机硬件特征解析

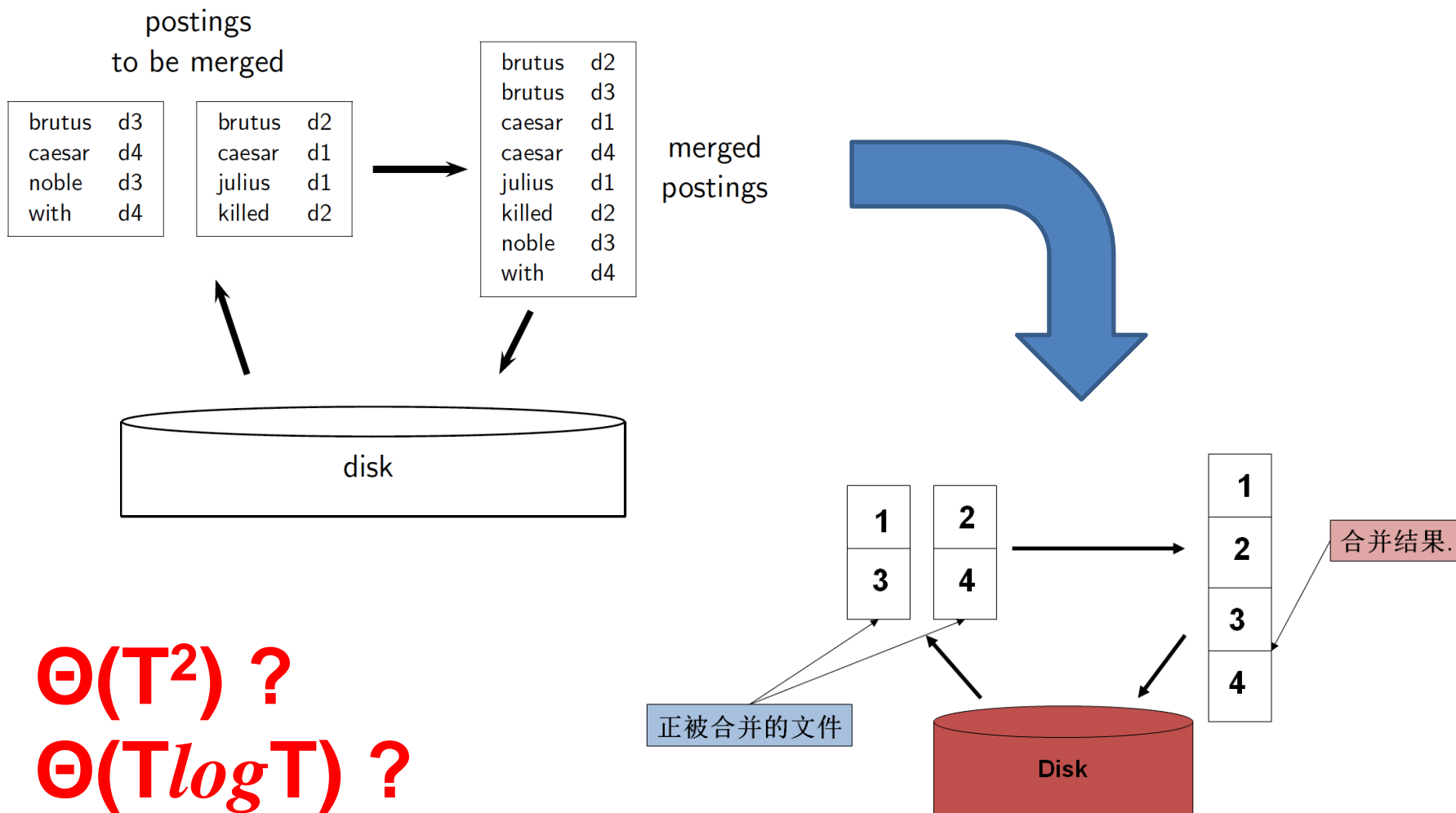


若硬盘**Cache 16MB**?

若16个硬盘的光通道阵列，顺序排列数据块?

回顾4-3: 索引构建算法 - 很简单的任务分解思想

BSBI(Blocked sort-based Indexing, 基于块的排序索引算法)



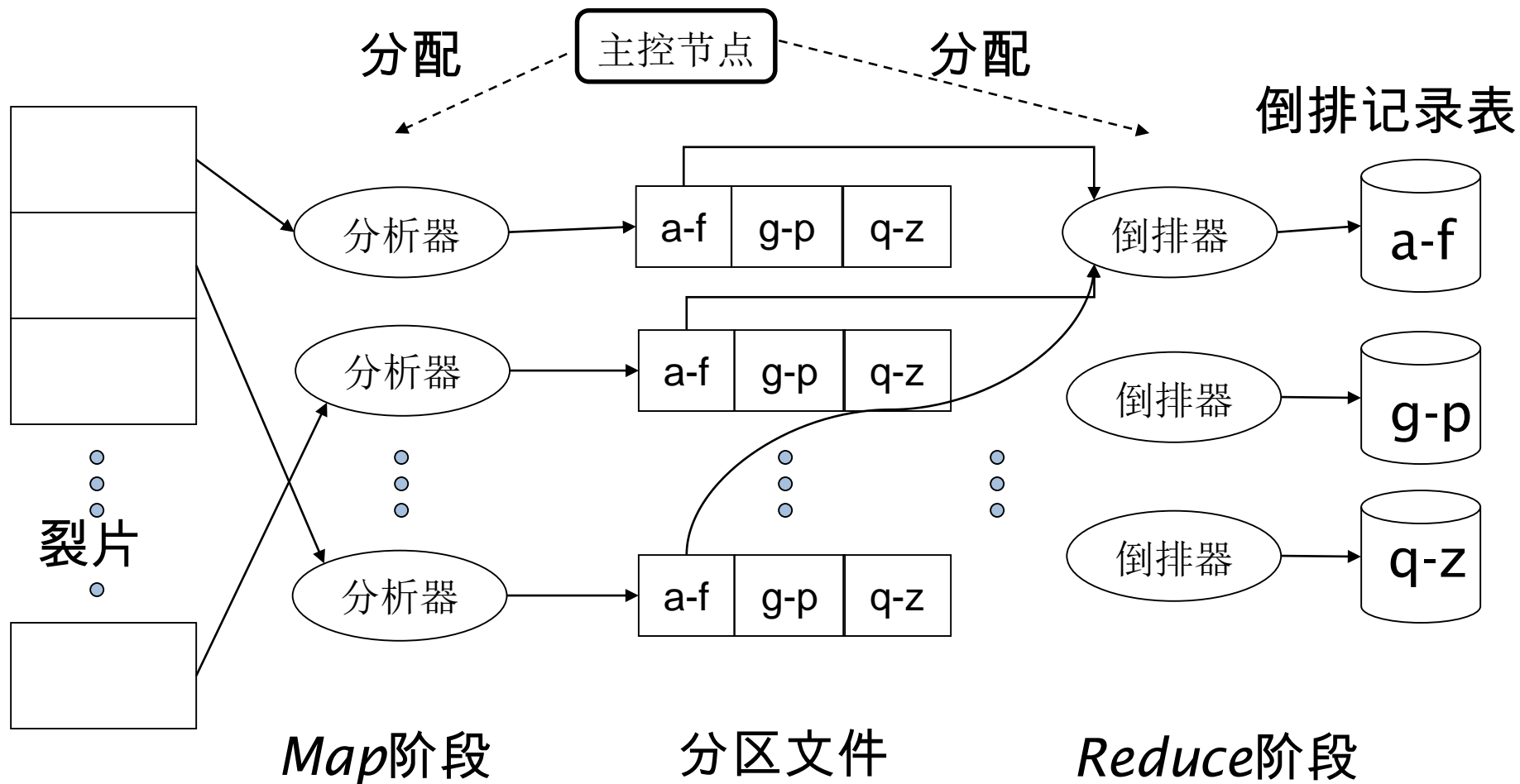
回顾4-4：索引构建算法 - 不做排序

SPIMI: Single-pass in-memory indexing, 内存式单遍扫描索引算法

- 核心思想1：为每个块单独生成一个词典——不需要维护全局的<词项，词项ID>映射表。
- 核心思想2：不进行排序。有新的<词项，文档ID>对时直接在倒排记录表中增加一项。
- 根据这两点思想，我们可以为每个块生成一个完整的倒排索引。
- 然后将这些单独的索引合并为一个大的索引。
- $\Theta(T)$

回顾4-5: 分布式索引构建- 基于词项分割

MapReduce进行分布式索引构建



回顾4-6：动态索引- 对数合并

- 文档集是静态 → 动态 → 索引要更新
- 最简单的方法
 - 维护一个大的主索引
 - 新文档信息存储在一个小的辅助索引中
- 对数合并
 - 维护一系列的索引，每个都是前一个的两倍大小。
 - 将最小的Z0存储在内存中
 - 较大的那些(I0, I1, ...)存储在磁盘中
 - 如果Z0达到上限n时，将它写入磁盘中，即成为I0
 - 当Z0下一次放满时，它会和I0合并，生成I1
 - Z1可以以I1的方式存储(如果I1不存在)
 - 或者和I1合并成I2(如果I1已存在)
 - ...

索引构建总结

- **基于排序的索引构建算法**
 - 它是一种最原始的在内存中进行倒排的方法
 - 基于块的排序索引算法
 - 合并排序操作对于基于磁盘的排序来说很高效(避免寻道)
- **内存式单遍扫描索引构建算法**
 - 没有全局的词典
 - 对每个块都生成单独的词典
 - 不对倒排记录进行排序
 - 有新的倒排记录出现时，直接在倒排记录表中增加一项
- **采用MapReduce的分布式索引构建算法**
- **动态索引构建算法：多个索引，对数合并**

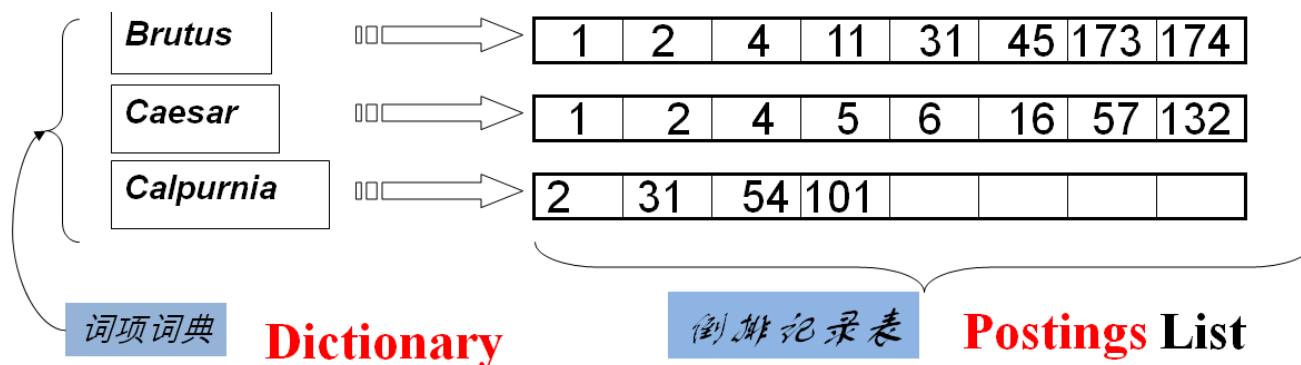
索引压缩

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?
- 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)	$\Delta\%$	T%	数目(K)	$\Delta\%$	T%	数目(K)	$\Delta\%$	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？
 - 实际上并不可行：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: $30 \leq k \leq 100$ 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 $\frac{1}{2}$ 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

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。

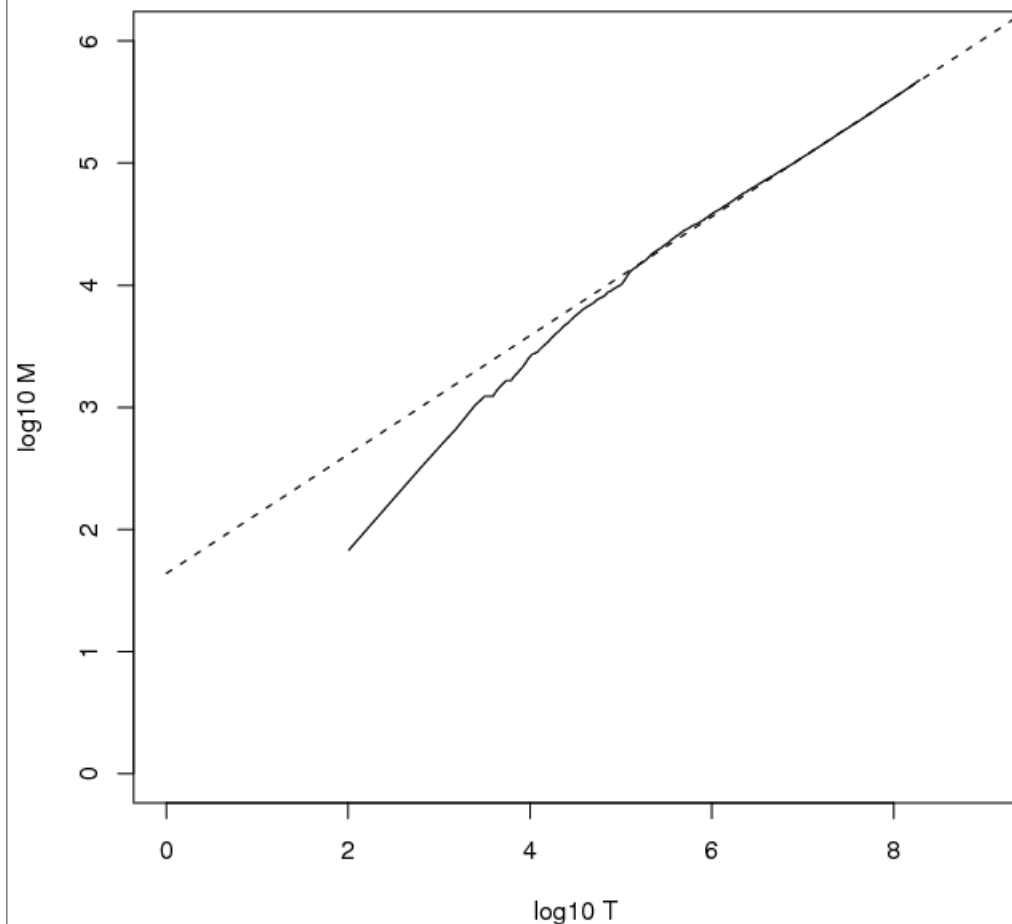
则 $M = 10^{1.64} T^{0.49}$ ，所以 $k = 10^{1.64} \approx 44$, $b = 0.49$

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



Zipf定律

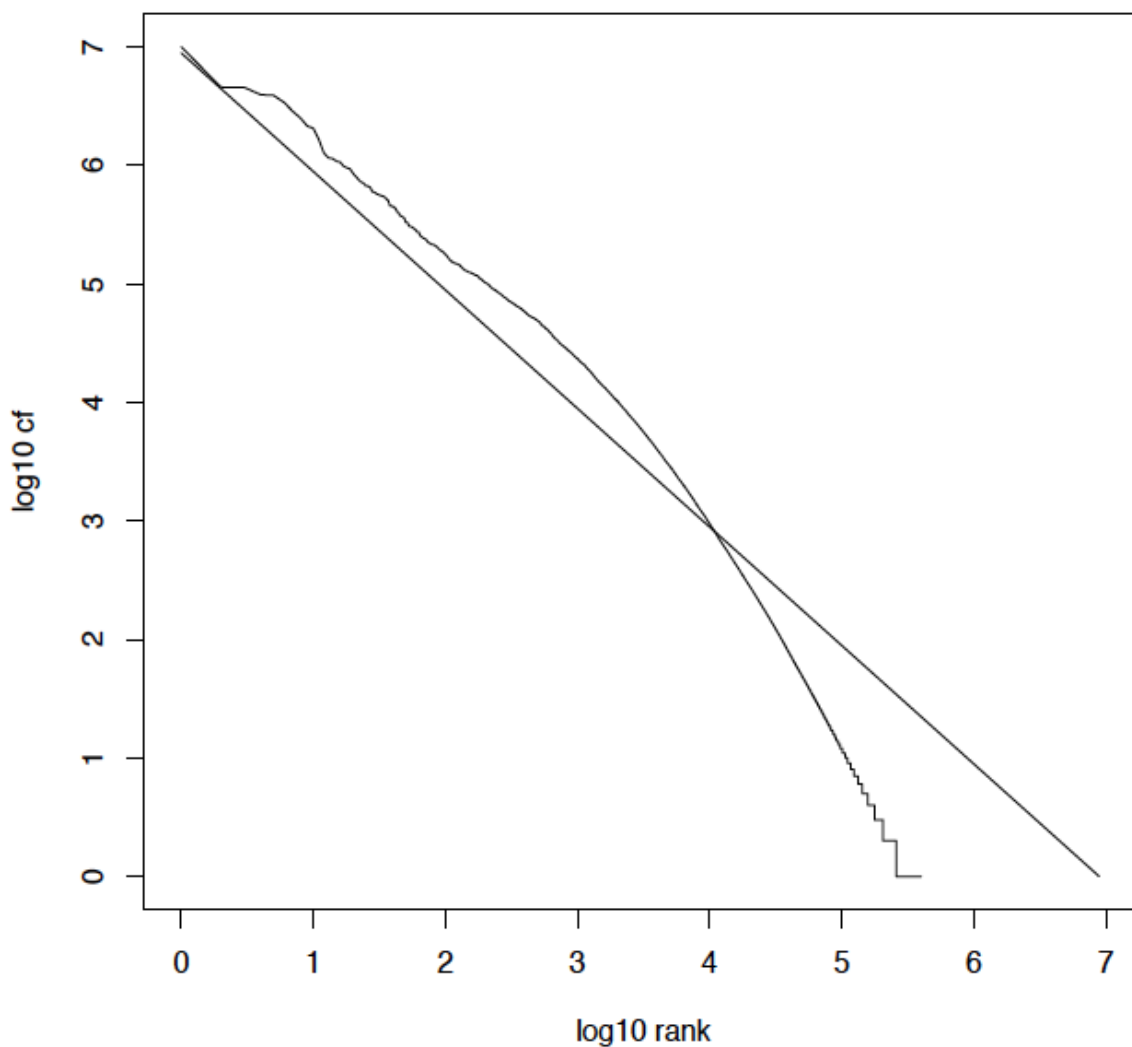
- Heaps定律提供了对文档集中词汇量的估计
- 我们还想了解词项在文档中的分布情况
- 在自然语言中，只有很少一些非常高频的词项，而其它绝大部分都是很生僻的词项。
- Zipf定律：排名第 i 多的词项的文档集频率与 $1/i$ 成正比
- $cf_i \propto 1/i = K/i$, K 是一个归一化常数
- Cf_i 是文档集频率：**词项** t_i 在文档集中出现的次数

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
 - $\log cf_i = \log K - \log i$
 - $\log cf_i$ 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



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.

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

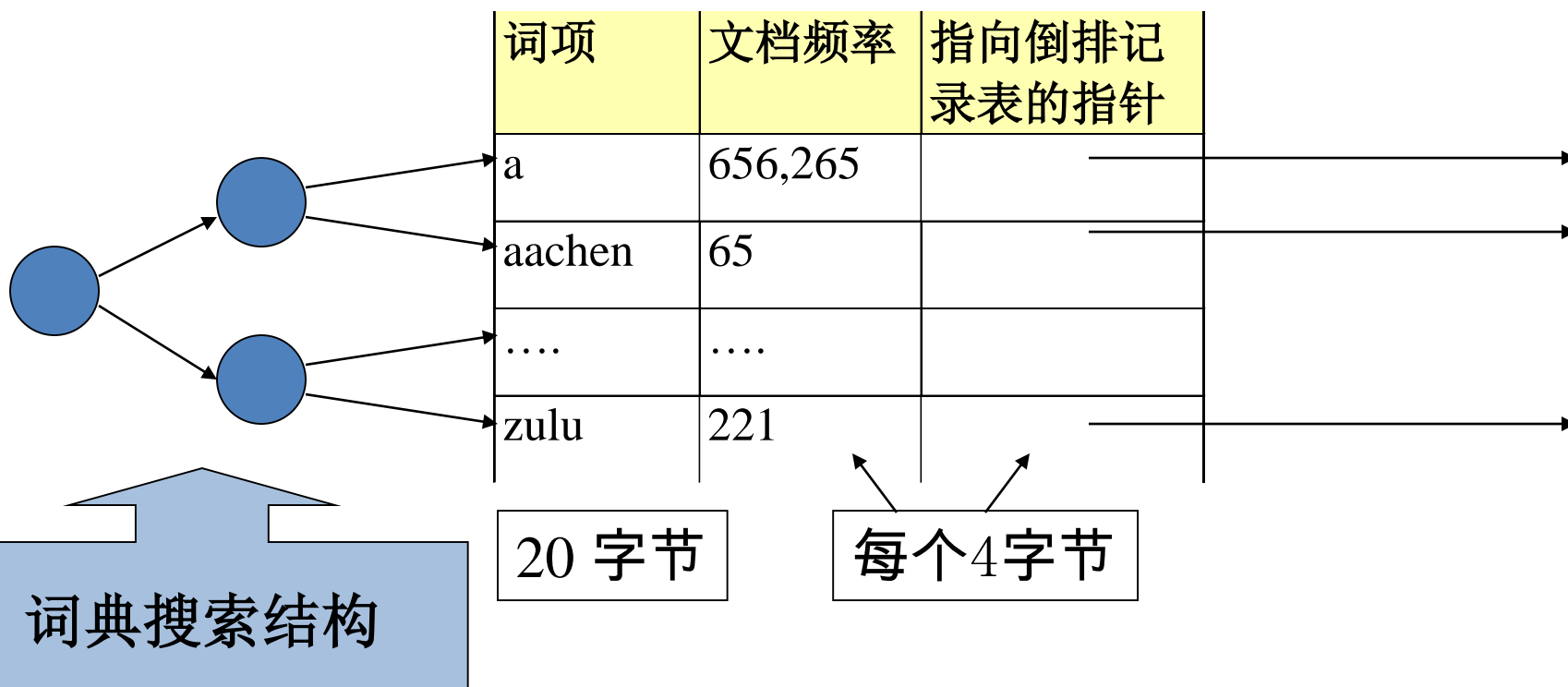
为什么要压缩词典？

- 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

词典存储

- 定长数组存储

- 400,000词项; 28字节/词项 = 11.2 MB

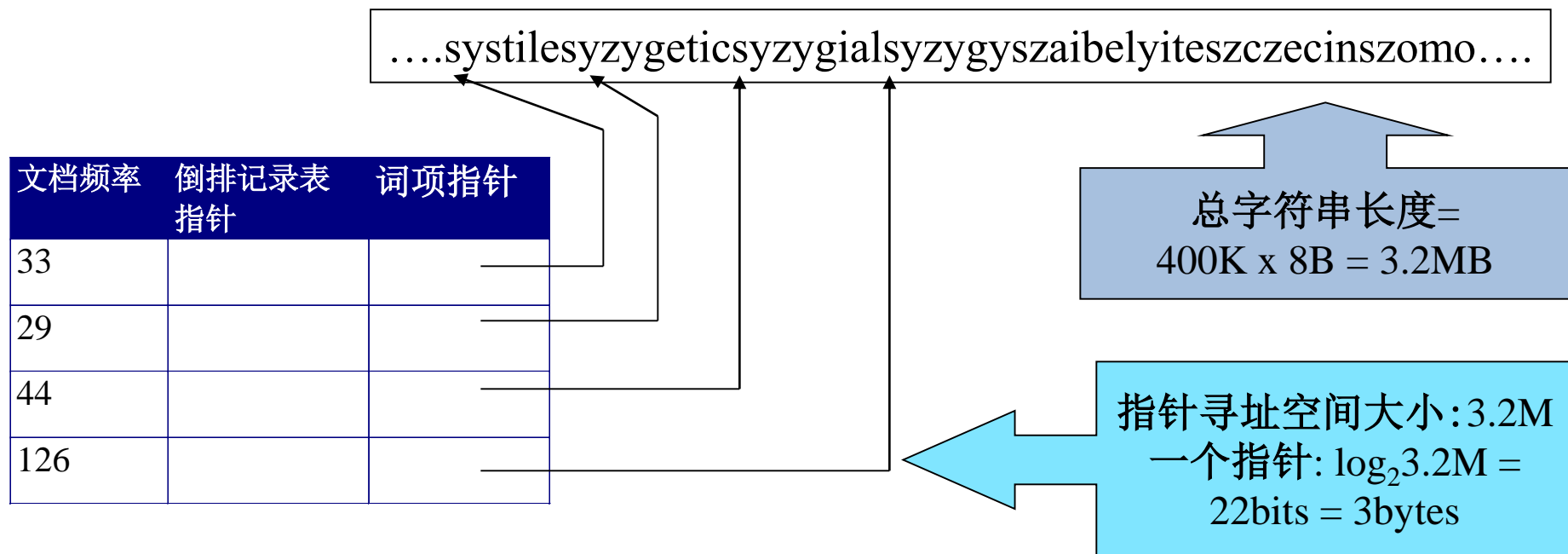


定长方法存储词项浪费空间

- 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

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

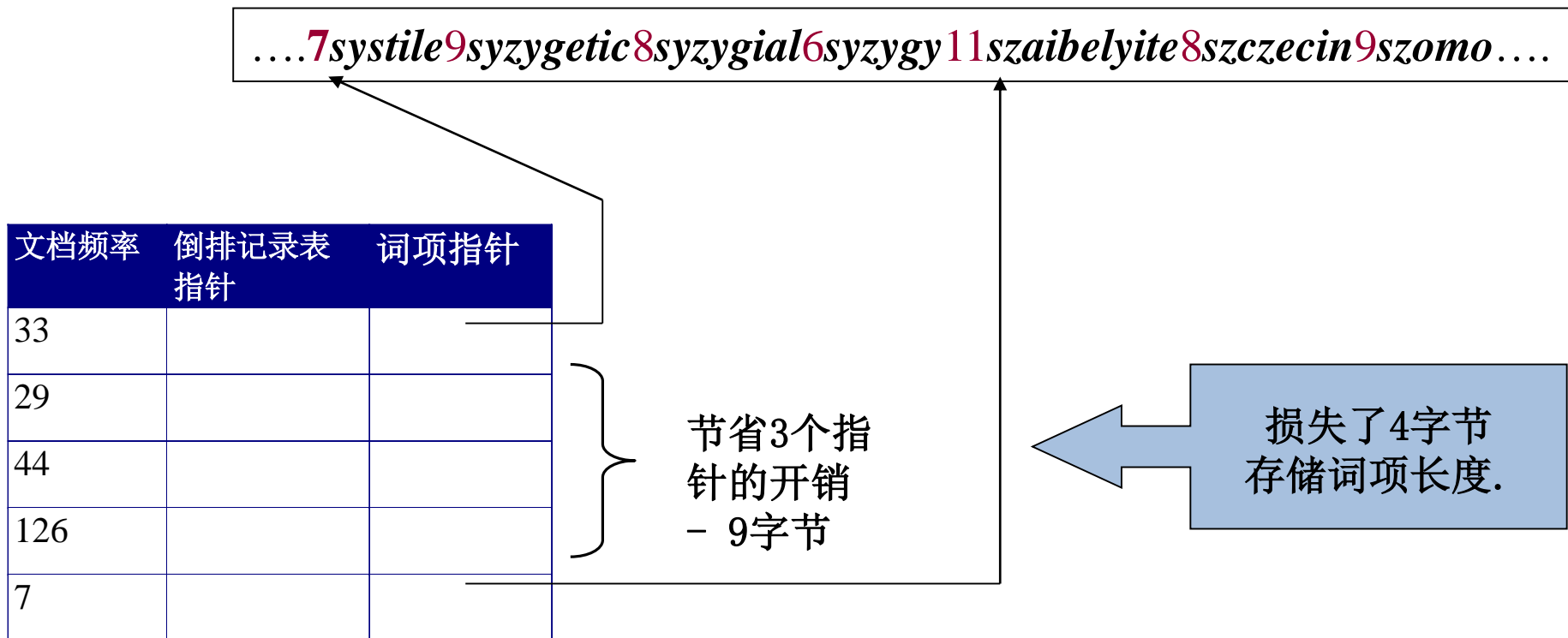


字符串词典的空间大小

- 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
 - $400k \text{词项} \times 19 = 7.6MB$ (固定长度时为 $11.2MB$)

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)



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
 - $3 \times 4 = 12$ 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：

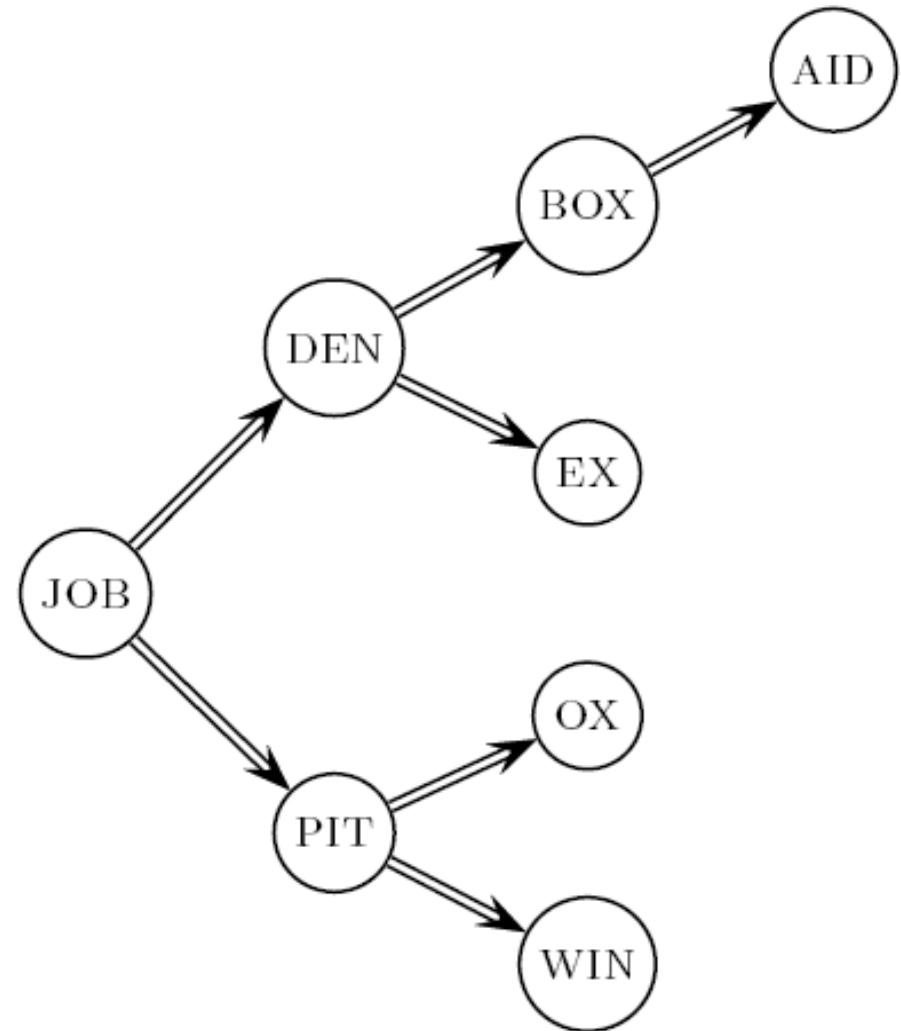
K=8 ?

K=16 ?

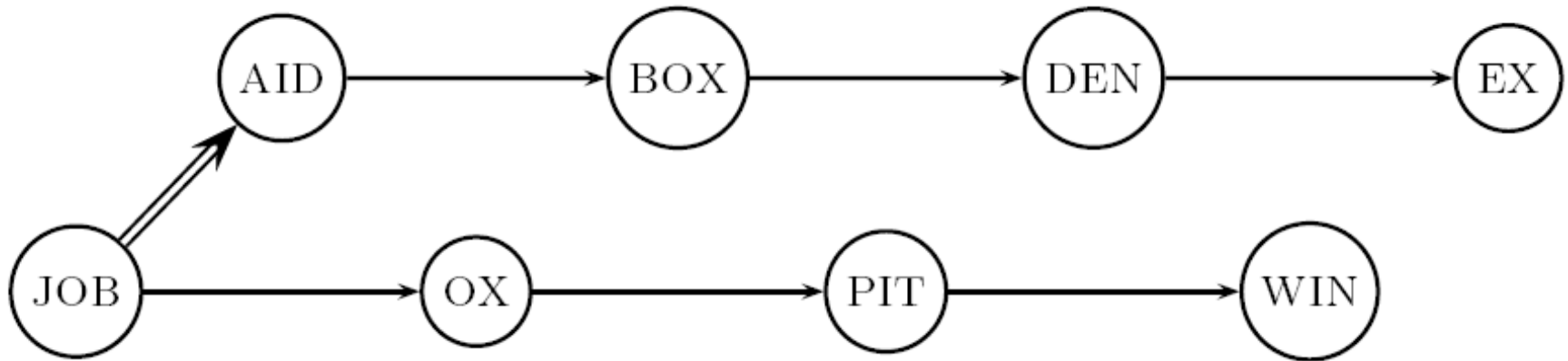
为什么不选取更大的k?

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)

8*automata***8***automate***9***automatic***10***automation*

→ **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 $\log_2 800,000 \approx 20 \text{ bits}$ 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

$$100M * 20 / 8 = 250MB$$

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)	$\Delta\%$	T%	数目(K)	$\Delta\%$	T%	数目(K)	$\Delta\%$	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 $\log_2 1M \sim 20$ bits 来存储这一倒排记录。
- 像 “*the*” 这样的词项在每个文档中都会出现，所以对它采用 20bit/倒排记录太浪费了
 - 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 *arachnocratic*, 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
- 先用一个字节来存储 G ，并分配1bit作为延续位 c
- 如果 $G \leq 127$ ，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 $c=0$

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

000001101011100010000101000011010000110010110001

824

5

214577

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 一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:

1110

- 80的一元编码为:

111

1110

- 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 \rightarrow 1101 \rightarrow 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 \rightarrow 1101=13

γ 编码特性

universal code



- G 的编码长度为 $2 \lfloor \log G \rfloor + 1$ bit
 - 偏移部分的编码长度是 $\lfloor \log G \rfloor$ bit
 - 长度部分的编码长度是 $\lfloor \log G \rfloor + 1$ 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，即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

谢谢大家!