信息检索导论

An Introduction to Information Retrieval

第5讲 索引压缩

Index compression

授课人：李波

中国科学院信息工程研究所/国科大网络空间安全学院

*改编自”An introduction to Information retrieval”网上公开的课件，地址 http://nlp.stanford.edu/IR-book/

提纲

1. 上一讲回顾
2. 压缩
3. 词项统计量
4. 词典压缩
5. 倒排记录表压缩

提纲

1. 上一讲回顾
3. 压缩
3. 词项统计量
3. 词典压缩
3. 倒排记录表压缩

第一讲中介绍的索引构建: 在内存中对倒排记录表进行排序(基于排序的索引构建方法)

- 第一步：对每篇文档，解析文档内容，抽取单词，生成单词-文档ID对

Doc 1

I did enact Julius Caesar I was killed i' the Capitol; Brutus killed me.

Doc 2

So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious

Term	Doc #
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

第一讲中介绍的索引构建: 在内存中对倒排记录表进行排序(基于排序的索引构建方法)

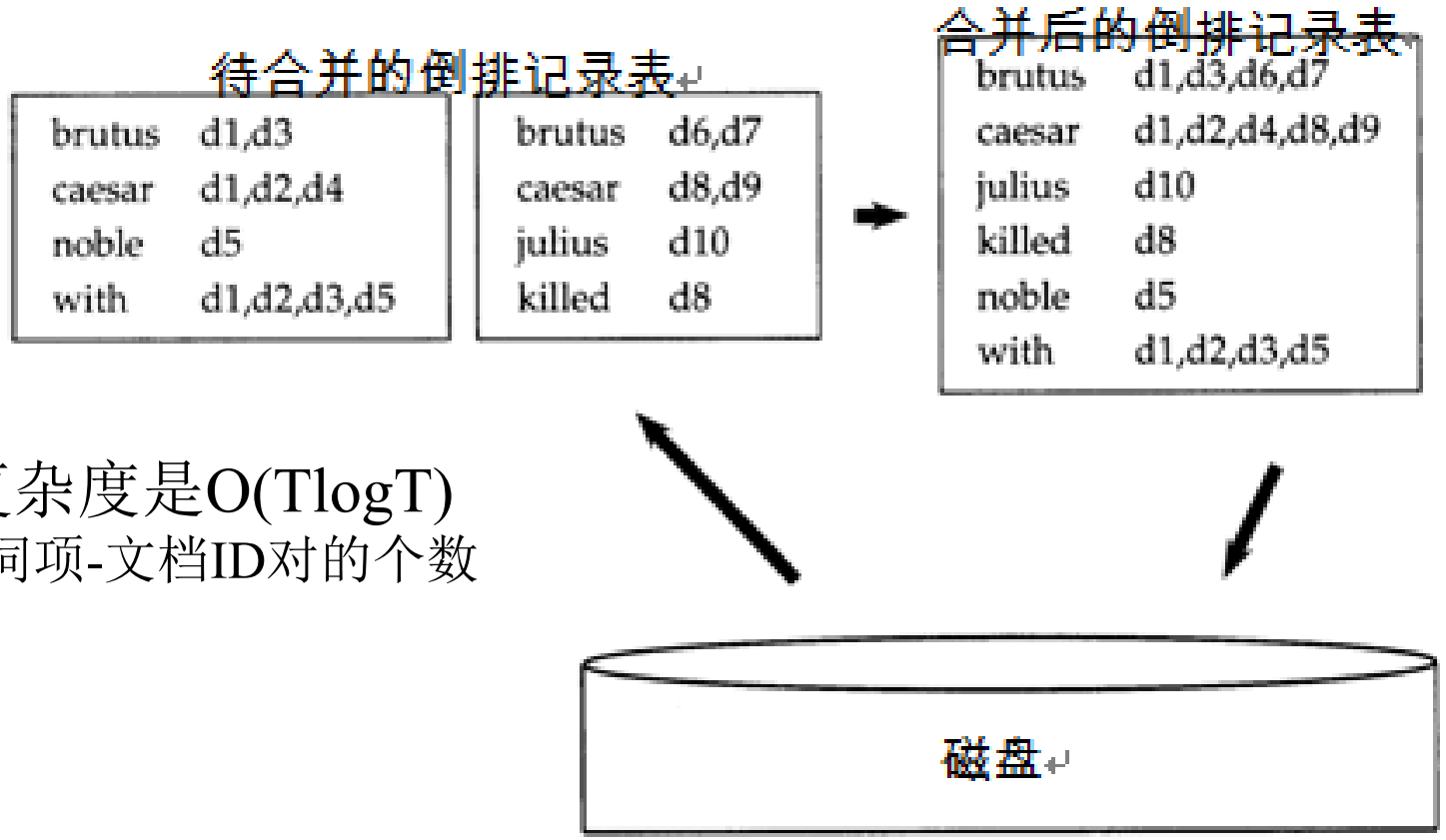
- 第二步：所有文档都解析完成后，按照词项进行排序，生成倒排索引

重点在该步的排序操作：
需要对100M个项进行排序！

Term	Doc #
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

Term	Doc #
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

基于排序的分块索引构建算法BSBI



时间复杂度是O(TlogT)

- T是词项-文档ID对的个数

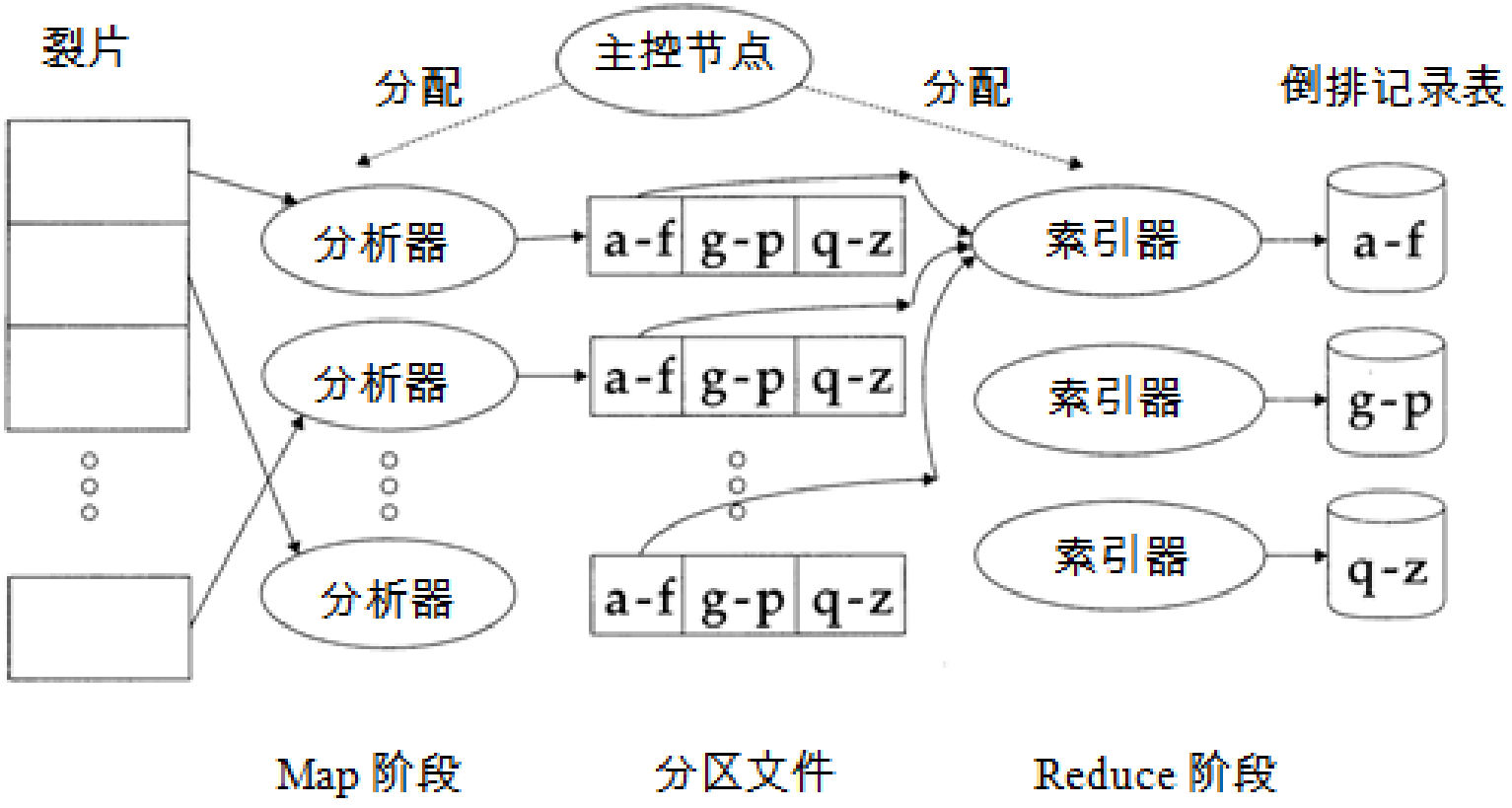
内存式单遍扫描索引构建算法SPIMI

- 关键思想 1: 对每个块都产生一个独立的词典
  - 不需要在块之间共享全局term-termID的映射
- 关键思想2: 按照词项出现的先后顺序分别构建索引
  - 中间过程无需按term/termID排序

- 基于上述思想可以对每个块生成一个完整的倒排索引
- 这些独立的索引最后合并成一个大索引

- 时间复杂度：O(T)

基于MapReduce的分布式索引构建



基于MapReduce的分布式索引构建

- 输入：
  - d1 : C came, C c'ed.
  - d2 : C died.
- Map:
  - <C,d1>, <came,d1>, <C,d1>, <c'ed, d1>, <C, d2>, <died,d2>
- Reduce:
  - (<C,(d1,d2,d1)>, <died,(d2)>, <came,(d1)>, <c'ed,(d1)>) →
  - (<C,(d1:2,d2:1)>, <died,(d2:1)>, <came,(d1:1)>, <c'ed,(d1:1)>)

Map阶段输出有多少个k-v对？

Reduce阶段输出的倒排记录数目是多少？

基于MapReduce的索引构建

- 上面的索引构建只是一个步骤，实现了按词项分割的索引
- 另一个步骤：将按词项分割的索引转换成按文档分割的索引
  - 按词项分割: 每个节点处理一部分词项索引
  - 按文档分割: 每个节点处理一部分文档集合
- 哪种分割方式更好？
  - 大多数搜索引擎使用文档分割索引
  - 具有更好的负载均衡

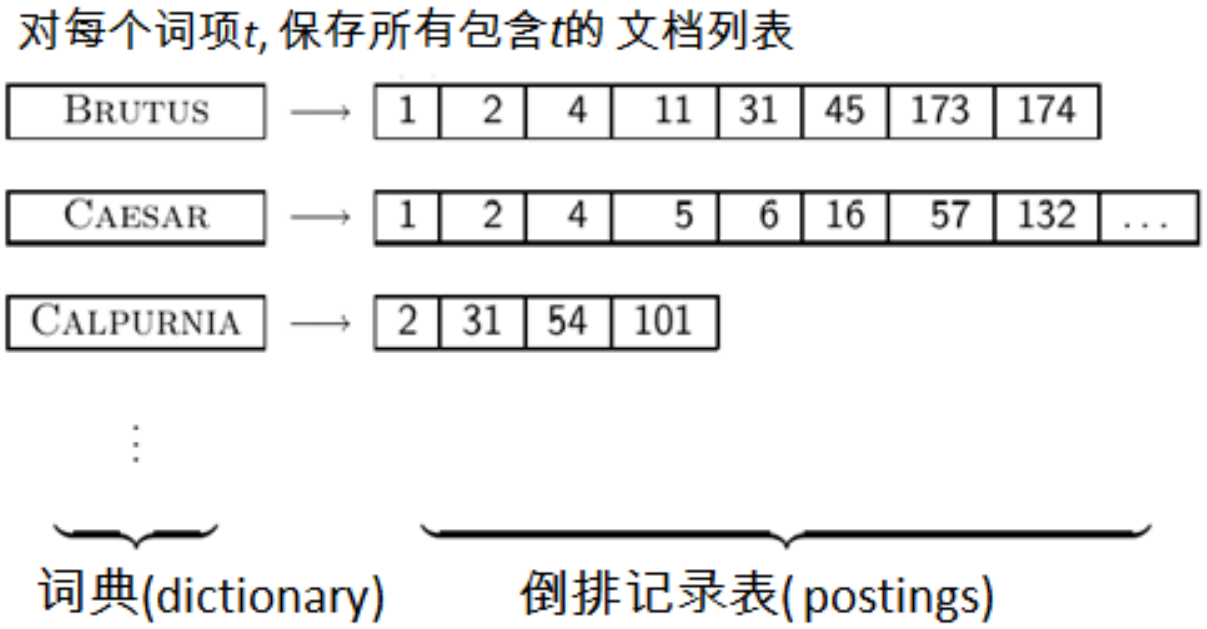
动态索引构建：最简单的方法

- 在磁盘上维护一个大的主索引(Main index)
- 新文档放入内存中较小的辅助索引(Auxiliary index)中
- 同时搜索两个索引，然后合并结果
- 定期将辅助索引合并到主索引中

思考题

- MapReduce能否应用于动态/实时索引构建？

本讲内容



- 信息检索中进行压缩的动机
- 词项统计量：词项在整个文档集中如何分布？
- 倒排索引中词典部分如何压缩？
- 倒排索引中倒排记录表部分如何压缩？

提纲

1. 上一讲回顾
2. 压缩
3. 词项统计量
4. 词典压缩
5. 倒排记录表压缩

Larger-Scale Computing

Google

Google的规模

- 全球最大规模的商业PC服务器集群
  - 超过1,000,000台服务器
- 索引的网页规模超过xx*10亿
  - 发现网页超过1万亿，但并不是所有的都建立索引
- 查询性能：
  - 每天处理2亿次搜索请求，平均在0.2秒返回结果（2003年）

- 如果索引不压缩，Google的查询响应时间是多少？

Q1: 100亿网页的大小是多少字节

- 仅考虑文本 ？

网页规模

- 约100亿网页
- 每个网页平均包含500个词条(ClueWeb09: ~900)
- 每个词条长度为5个字符
- 存储这些网页需要：
  - 10^{10} x 500 x 6 ~ 30 TB

Q2: 扫描一遍这30TB数据需要多少时间？

- 如何估算？

搜索30TB数据需要多长时间？

- 需要一块大容量硬盘
  - 大小: 30 TB ??
  - 硬盘传输速度：100 MB/s
- 顺序扫描所需的时间：
  - 300,000 秒
  - 响应1个请求，需要3.5天
- 如何查询更快？

倒排索引


BRUTUS
1 2 4 11 31 45 173 174
CAESAR
1 2 4 5 6 16 57 132 . . .
CALPURNIA
2 31 54 101
词典(dictionary)
倒排记录表(postings)

Q3: 估算一下倒排索引的大小

?

忽略如下因素

- 文本统计信息:
  - 词频
  - 文档频率等
- 超链接统计信息:
  - 链入/链出信息
  - 锚文本等
- 词项位置信息

倒排索引大小估算

- 倒排记录的大小(词项-文档对):
  - 文档数目: ~10^{10}
- 每个文档中出现的独立词项数目（文档词条数为500）： ~250
- 每个倒排记录项需要5字节（为什么？）
- 倒排表大小：10^{10} x 250 x 5 = 12.5 TB
- 倒排表的大小约为数据大小的50%

倒排索引大小估算

- 词项数量：10^8
  - 平均长度6字节
  - 加上词频、倒排记录的指针，需要8字节
- 字典表大小：10^8 x 14 = 1.4 GB
  - 相对倒排记录来说小得多 (0.01 %)

Q4: 使用倒排索引查询“information retrieval”需要多长时间？

- 假设两个词项的选择率：
  - Information在10亿（10^9）个网页中出现
  - retrieval 在1千万（10^7）个网页中出现

查询处理流程（1）

- 每个词项的倒排记录是升序的整数序列
  - 两个倒排记录的合并时间与两个整数序列的长度之和成正比

查询处理（2）

- 查询通常被认为是合取查询
- 查询请求：information retrieval实际上是
  - information AND retrieval
- 假设
  - [<retrieval; 7; [2, 23, 81, 98, 121, 126, 139]>
  - [<information; 9; [1, 14, 23, 45, 46, 84, 98, 111, 120]>
- 两个倒排记录表的交集：
  - [23, 98]

查询处理流程（3）

- 回顾一下布尔模型？
- 对倒排记录求交集、并集和补集
- 对于information OR retrieval
  - [<retrieval; 7; [2, 23, 81, 98, 121, 126, 139]>
  - [<information; 9; [1, 14, 23, 45, 46, 84, 98, 111, 120]>
- 两个倒排记录表的并集：
  - [1, 2, 14, 23, 45, 46, 81, 84, 98, 111, 120, 121, 126, 139]

查询处理流程（4）

- 估算每个词项的选择率
  - 假设information出现在10^9个网页中
  - Retrieval出现在10^7个网页中
- 倒排记录表的大小（每个docid占5字节）
  - information：$10^9 * 5B = 5\ GB$
  - retrieval：$10^7 * 5B = 50\ MB$
- 磁盘读取倒排记录的时间:
  - 50秒读取information + 0.5秒读取retrieval
- 忽略CPU耗时和磁盘延迟

查询处理流程（5）

- 查询响应时间由3天降低到50.5秒！
- :-)

- 但是... 仍然太慢...
- :-(

什么是压缩？

- 将长编码串用短编码串来代替
  - 11111111111111111➔ 18个1

为什么要压缩? (一般意义上而言)

- 减少磁盘空间 (节省开销)
- 增加内存缓存内容 (加快速度)
- 加快从磁盘到内存的数据传输速度 (同样加快速度)
  - [读压缩数据到内存+在内存中解压]比直接读入未压缩数据要快很多
  - 前提: 解压速度很快
- 本讲我们介绍的解压算法的速度都很快

为什么在IR中需要压缩?

- 首先，需要考虑词典的存储空间
  - 词典压缩的主要动机: 使之能够尽量放入内存中
- 其次，对于倒排记录表而言
  - 动机: 减少磁盘存储空间，减少从磁盘读入内存的时间
  - 注意: 大型搜索引擎将相当比例的倒排记录表都放入内存
- 接下来，将介绍词典压缩和倒排记录表压缩的多种机制

有损(Lossy) vs. 无损(Lossless)压缩

- **有损压缩**: 丢弃一些信息
- 前面讲到的很多常用的预处理步骤可以看成是有损压缩:
  - 统一小写,去除停用词，Porter词干还原，去掉数字

- **无损压缩**: 所有信息都保留
  - 索引压缩中通常都使用无损压缩

提纲

词典压缩和倒排记录表压缩

- 词典压缩中词典的大小即词汇表的大小是关键
  - 能否预测词典的大小？
- 倒排记录表压缩中词项的分布情况是关键
  - 能否对词项的分布进行估计？
- 引入词项统计量对上述进行估计，引出两个经验法则

对文档集建模： Reuters RCV1

N	文档数目	800,000
L	每篇文档的词条数目	200
M	词项数目(= 词类数目)	400,000
	每个词条的字节数 (含空格和标点)	6
	每个词条的字节数 (不含空格和标点)	4.5
	每个词项的字节数	7.5
T	无位置信息索引中的倒排记录数目	100,000,000

预处理的效果

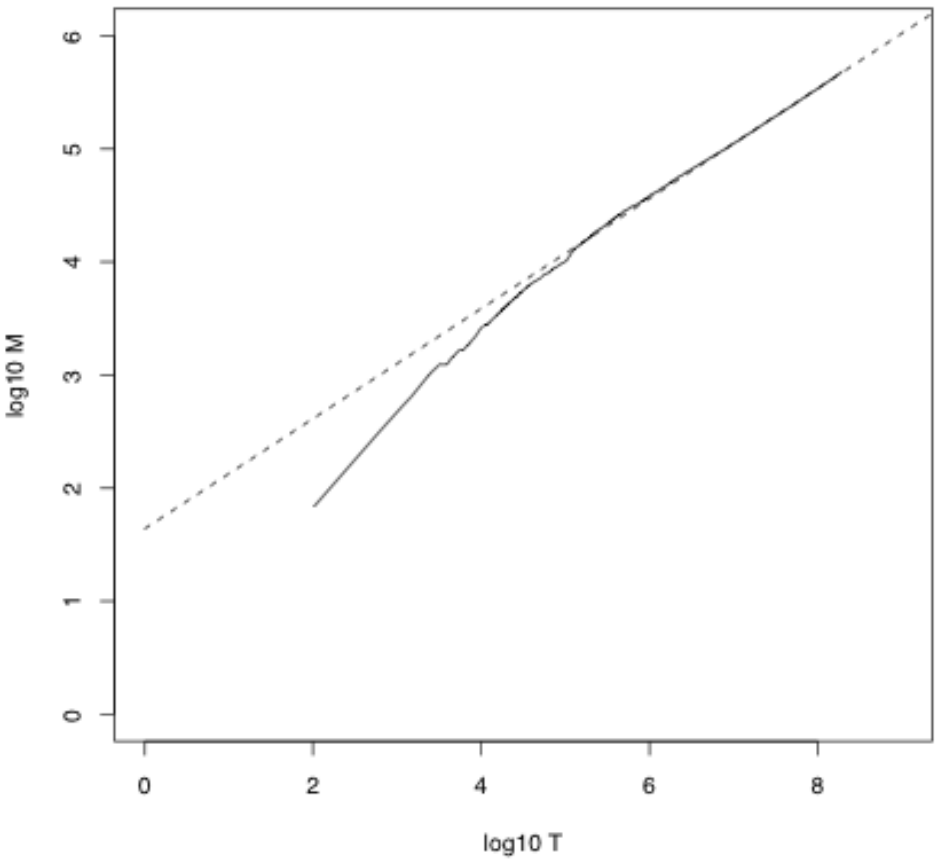
	不同词项			无位置信息倒排记录			词　条①		
	数目	Δ%	T%	数目	Δ%	T%	数目	Δ%	T%
未过滤	484 494			109 971 179			197 879 290		
无数字	473 723	–2	–2	100 680 242	–8	–8	179 158 204	–9	–9
大小写转换	391 523	–17	–19	96 969 056	–3	–12	179 158 204	–0	–9
30个停用词	391 493	–0	–19	83 390 443	–14	–24	121 857 825	–31	–38
150个停用词	391 373	–0	–19	67 001 847	–30	–39	94 516 599	–47	–52
词干还原	322 383	–17	–33	63 812 300	–4	–42	94 516 599	–0	–52

- 不同词条化方法得到的未压缩索引大小是多少？

第一个问题：词汇表有多大(即词项数目)?

- 即有多少不同的单词数目?
  - 首先，能否假设这个数目存在一个上界?
  - 不能：对于长度为20的单词，有大约$70^{20} \approx 10^{37}$ 种可能的单词
- 实际上，词汇表大小会随着文档集的大小增长而增长!
- **Heaps定律:** $M = kT^b$
- M 是词汇表大小, T 是文档集的大小(所有词条的个数，即所有文档大小之和)
- 参数k 和b 的一个经典取值是: $30 \leq k \leq 100$ 及 $b \approx 0.5$.
- **Heaps定律在对数空间下是线性的**
  - 这也是在对数空间下两者之间最简单的关系
  - 经验规律

Reuters RCV1上的Heaps定律



- 词汇表大小M是文档集规模T的一个函数
- 图中通过最小二乘法拟合出的直线方程为：

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

- 于是有：
- $M = 10^{1.64} T^{0.49}$
- $k = 10^{1.64} \approx 44$
- $b = 0.49$

拟合 vs. 真实

- 例子: 对于前1,000,020个词条， 根据Heaps定律预计将有38,323个词项:

$$44 \times 1{,}000{,}020^{0.49} \approx 38{,}323$$

- 实际的词项数目为38,365，和预测值非常接近
- 经验上的观察结果表明，一般情况下拟合度还是非常高的

课堂练习

1. 在容许拼写错误或者对拼写错误自动纠错的情况下，Heaps定律的效果如何？
2. 计算词汇表大小 M
   - 观察一个网页集合，你会发现在前10000个词条中有3000个不同的词项，在前1000000个词条中有30000个不同的词项
   - 假定某搜索引擎索引了总共20,000,000,000 (2×10^{10})个网页, 平均每个网页包含200个词条
   - 那么按照Heaps定律，被索引的文档集的词汇表大小是多少？

$$M = 44 * T^{0.49}$$

第二个问题：词项的分布如何？Zipf定律

- Heaps定律告诉我们随着文档集规模的增长词项的增长情况
- 但是我们还需要知道在文档集中有多少高频词项 vs. 低频词项。
- 在自然语言中，有一些极高频词项，有大量极低频的罕见词项
- Zipf定律: 第i常见的词项的频率cf_i和$1/i$成正比
- $\mathrm{cf}_i \propto \frac{1}{i}$
- cf_i是文档集频率(collection frequency): 词项t_i在所有文档中出现的次数(不是出现该词项的文档数目df)

Zipf定律

Zipf's law: Rank × Frequency ∼ Constant

English:	Rank R	Word	Frequency f	$R \times f$
	10	he	877	8770
	20	but	410	8200
	30	be	294	8820
	800	friends	10	8000
	1000	family	8	8000

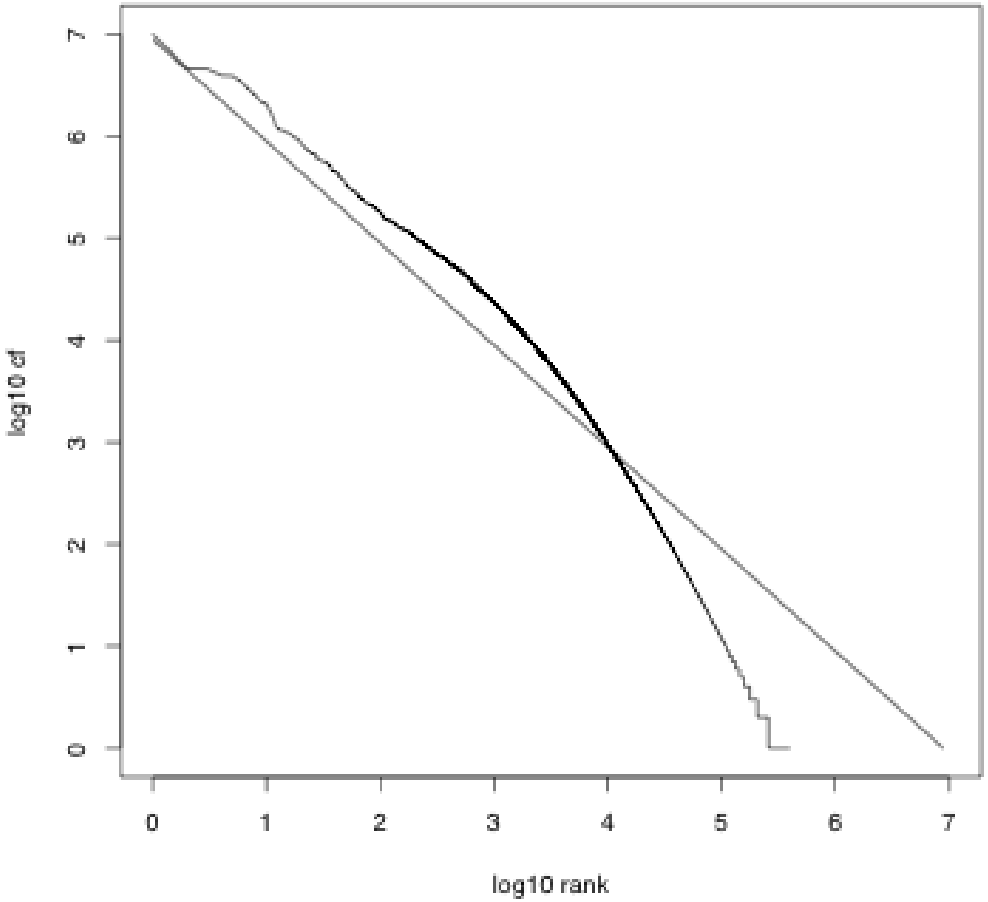
German:	Rank R	Word	Frequency f	$R \times f$
	10	sich	1,680,106	16,801,060
	100	immer	197,502	19,750,200
	500	Mio	36,116	18,059,500
	1,000	Medien	19,041	19,041,000
	5,000	Miete	3,755	19,041,000
	10,000	vorläufige	1.664	16,640,000

*本页借用了Ronan Cummins的课件

Zipf定律

- Zipf定律: 第*i*常见的词项的频率cf_i和1/*i* 成正比
- $\mathrm{cf}_i \propto \frac{1}{i}$
- cf_i 是文档频率(collection frequency): 词项t_i在所有文档中出现的次数(不是出现该词项的文档数目df).
- 于是，如果最常见的词项(*the*)出现cf_1 次，那么第二常见的词项 (*of*)出现次数 $\mathrm{cf}_2 = \frac{1}{2}\mathrm{cf}_1 \quad \ldots$
- . . . 第三常见的词项 (*and*) 出现次数为 $\mathrm{cf}_3 = \frac{1}{3}\mathrm{cf}_1$
- 另一种表示方式: $\mathrm{cf}_i = ci^k$ 或 $\log \mathrm{cf}_i = \log c + k \log i \quad (k = -1)$
- 幂定律(power law)的一个实例

Reuters RCV1上Zipf定律的体现



拟合度不是非常高，但是最重要的是如下关键性发现：高频词项很少，低频罕见词项很多

提纲

词典压缩

- 一般而言，相对于倒排记录表，词典所占空间较小
- 但是我们想将词典放入内存
- 另外，满足一些特定领域特定应用的需要，如手机、机载计算机上的应用或要求快速启动等需求
- 因此，压缩词典相当重要

回顾: 定长数组方式下的词典存储

词项	文档频率	指向倒排记录表的指针
a	656 265	→
aachen	65	→
…	…	…
zulu	221	→

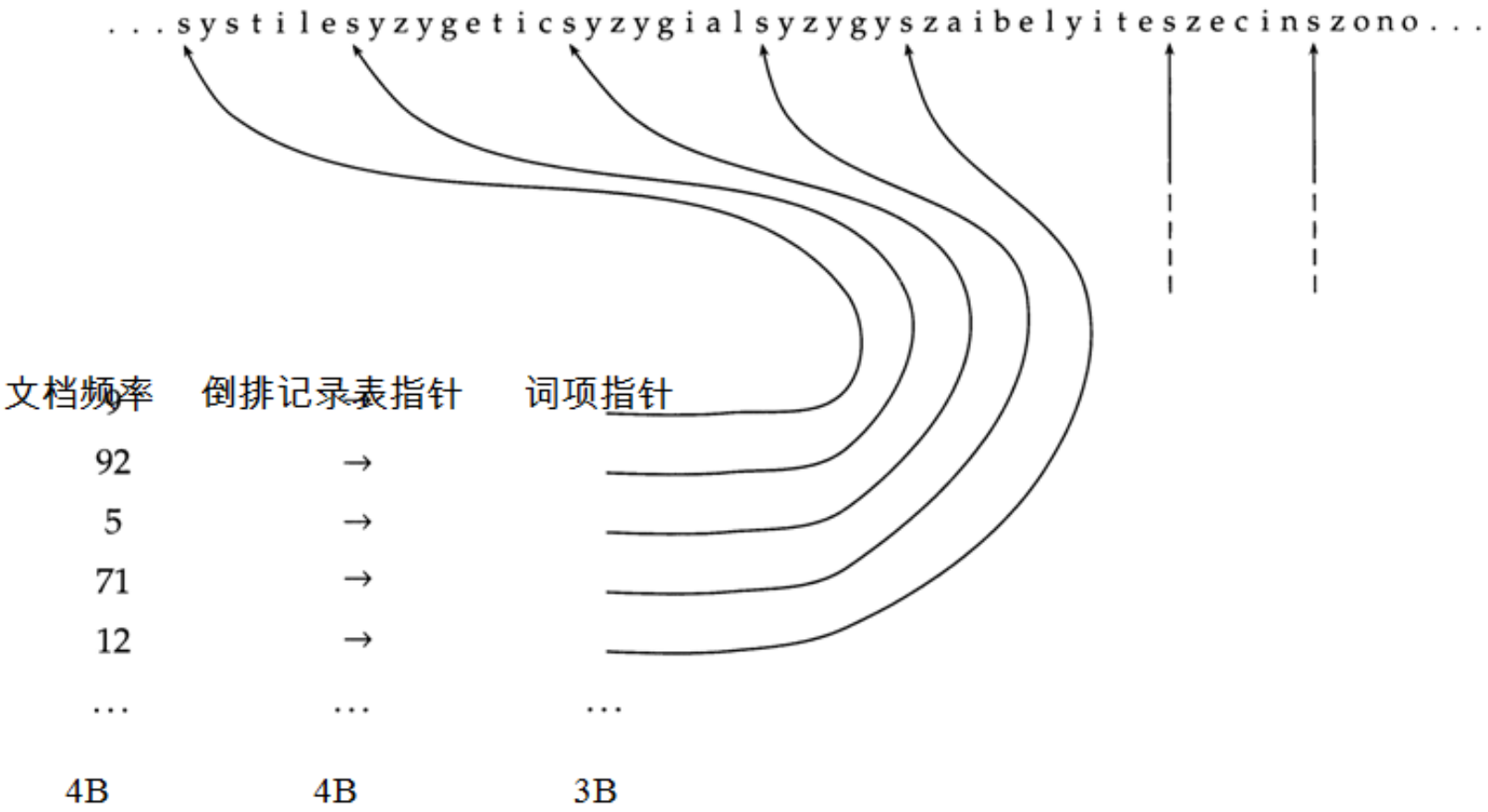
空间需求： 20 字节 4 字节 4 字节

对Reuters RCV1语料: (20+4+4)*400,000 = 11.2 MB

定长方式的不足

- 大量存储空间被浪费
  - 即使是长度为1的词项，我们也分配20个字节
- 不能处理长度大于20字节的词项，如HYDROCHLOROFLUOROCARBONS 和 SUPERCALIFRAGILISTICEXPIALIDOCIOUS
- 而英语中每个词项的平均长度为8个字符
- 能否对每个词项平均只使用8个字节来存储?

将整部词典看成单一字符串(Dictionary as a string)



单一字符串方式下的空间消耗

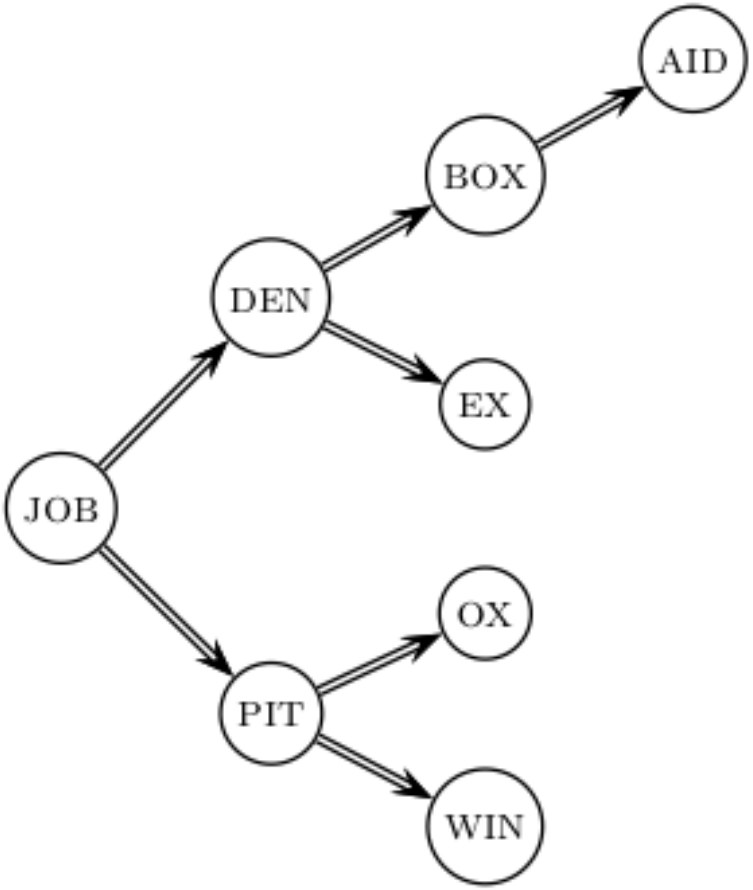
- 每个词项的词项频率需要4个字节
- 每个词项指向倒排记录表的指针需要4个字节
- 每个词项平均需要8个字节
- 指向字符串的指针需要3个字节 (8*400000个位置需要 $\log_2$（8 * 400000）＜24 位来表示)
- 空间消耗: 400,000 × (4 +4 +3 + 8) = 7.6MB (而定长数组方式需要11.2MB)

单一字符串方式下按块存储

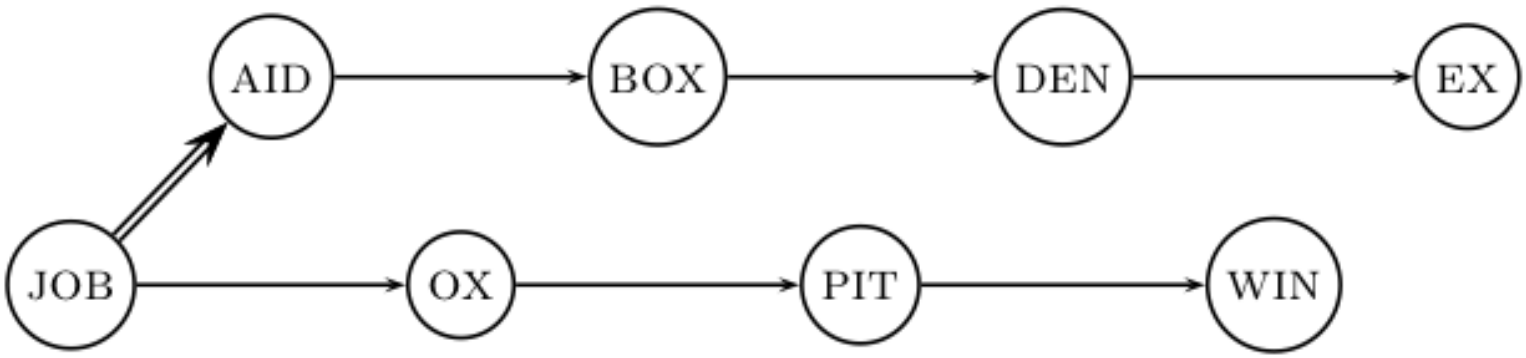
按块存储下的空间消耗

- 如果不按块存储，则每4个词项指针将占据空间4 × 3=12B
- 现在按块存储，假设块大小k=4，此时每4个词项只需要保留1个词项指针，但是同时需要增加4个字节来表示每个词项的长度，此时每4个词项需要3+4=7B
- 因此，每4个词项将节省12-7=5B
- 于是，整个词典空间将节省40,000/4*5B=0.5MB
- 最终的词典空间将从7.6MB压缩至7.1MB

不采用块存储方式下的词项查找



采用块存储方式下的词项查找: 稍慢



前端编码(Front coding)

- 每个块当中 (k = 4)，会有公共前缀 . . .
- 8automata8automate9automatic10automation
- ⇓
- . . . 可以采用前端编码方式继续压缩
- 8automat∗a1⋄e2⋄ic3⋄ion

Reuters RCV1词典压缩情况总表

数据结构	压缩后的空间大小（单位：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

课堂练习

- 哪些前缀应该用于前端编码？需要在哪些方面有所权衡？
- 输入: 词项列表，即词汇表
- 输出: 用于前端编码的前缀列表

提纲

倒排记录表压缩

- 倒排记录表空间远大于词典，至少10倍以上
- 压缩关键：对每条倒排记录进行压缩
- 目前每条倒排记录表中存放的是docID.
- 对于Reuters RCV1(800,000篇文档), 当每个docID可以采用4字节（即32位）整数来表示
- 当然，我们也可以采用$\log_2 800{,}000 \approx 19.6 < 20$ 位来表示每个docID
- 我们的压缩目标是： 压缩后每个docID用到的位数远小于20比特

关键思想: 存储docID间隔而不是docID本身

- 每个倒排记录表中的docID是从低到高排序
  - 例子: COMPUTER: 283154, 283159, 283202, . . .
- 存储间隔能够降低开销：283159-283154=5, 283202-283159=43
- 于是可以顺序存储间隔(第一个不是间隔) : COMPUTER: 283154, 5, 43, . . .
- 高频词项的间隔较小
- 因此，可以对这些间隔采用小于20比特的存储方式

对间隔编码

	编码对象	倒排记录表					
the	文档ID	…	283 042	283 043	283 044	283 045	…
	文档ID间距			1	1	1	…
computer	文档ID	…	283 047	283 154	283 159	283 202	…
	文档ID间距			107	5	43	…
arachnocentric	文档ID	252 000	500 100				
	文档ID间距	252 000	248 100				

对间隔编码

- 方法1：
- 由存储doc-id列表：
  - [<retrieval; 7; [2, 23, 81, 98, 121, 126, 180]>
- 改为存储间隔的列表
  - [<retrieval; 7; [2, 21, 58, 17, 23, 5, 54]>

- 没有信息损失
- 总是从头处理倒排记录表，解码简单

对间隔编码

- 存在的问题？
  - 间隔的最大值取决于索引的网页总数
  - 不频繁词项的间隔较少/值较大
  - 频繁词项的间隔较多/值较小

- 方法2：对间隔进行变长编码

变长编码

- 目标:
  - 对于 ARACHNOCENTRIC 及其他罕见词项, 对每个间隔仍然使用20比特
  - 对于THE及其他高频词项，每个间隔仅仅使用很少的比特位来编码
- 为了实现上述目标，需要设计一个变长编码(variable length encoding)
- 可变长编码对于小间隔采用短编码而对于长间隔采用长编码

可变字节(VB)码

- 被很多商用/研究系统所采用
- 变长编码及对齐敏感性(指匹配时按字节对齐还是按照位对齐)的简单且不错的混合产物
- 设定一个专用位 (高位) *c*作为延续位(continuation bit)
- 如果间隔表示少于7位，那么*c* 置 1，将间隔编入一个字节的后7位中
- 否则：将高7位放入当前字节中，并将*c* 置 0，剩下的位数采用同样的方法进行处理，最后一个字节的*c*置1（表示结束）
  - 将当前二进制按7的整数倍补齐位数（高位补0）
  - 从高到低编码

VB 编码的例子

文档ID	824	829	215 406
间距		5	214 577
VB编码	00000110 10111000	10000101	00001101 00001100 10110001

VB 编码算法

```
VBENCODENUMBER(n)
1  bytes ← ⟨⟩
2  while true
3  do PREPEND(bytes, n mod 128)
4     if n < 128
5        then BREAK
6     n ← n div 128
7  bytes[LENGTH(bytes)] += 128
8  return bytes
```

```
VBENCODE(numbers)
1  bytestream ← ⟨⟩
2  for each n ∈ numbers
3  do bytes ← VBENCODENUMBER(n)
4     bytestream ← EXTEND(bytestream, bytes)
5  return bytestream
```

130 mod 128 = 2 --> bytes 数组
130 div 128 = 1, preappend 到 bytes数组

于是循环结束有 bytes=[1,2]
算法最后一步，是在bytes[length(bytes)]上加128

VB编码的解码算法

```
VBDecode(bytestream)
 1  numbers ← ⟨⟩
 2  n ← 0
 3  for i ← 1 to Length(bytestream)
 4  do if bytestream[i] < 128
 5        then n ← 128 × n + bytestream[i]
 6        else n ← 128 × n + (bytestream[i] − 128)
 7             Append(numbers, n)
 8             n ← 0
 9  return numbers
```

其它编码

- 除字节外，还可以采用不同的对齐单位：比如32位(word)、16位及4位(nibble)等等
- 如果有很多很小的间隔，那么采用可变字节码会浪费很多空间，而此时采用4位为单位将会节省空间
- 最近一些工作采用了32位的方式– 参考讲义末尾的参考材料

γ编码

- 另外一种变长编码是基于位的编码
- γ 编码是其中最出名的一种
- 首先，在介绍γ编码之前先介绍一元码(unary code)
- 一元码：
  - 将 n 表示成 n 个1和最后一个0
  - 比如： 3的一元码是 1110
  - 40的一元码是 1111111111111111111111111111111111111110
  - 70的一元码是：

1111111111111111111111111111111111111111111111111111111111111110

γ编码

- 将G 表示成长度(length)和偏移(offset)两部分
- 偏移对应G的二进制编码，只不过将首部的1去掉
- 例如 13 → 1101 → 101 = 偏移
- 长度部分给出的是偏移的位数
- 比如G=13 (偏移为 101), 长度部分为 3
- 长度部分采用一元编码: 1110.
- 于是G的γ编码就是将长度部分和偏移部分两者联接起来得到的结果。

γ编码的例子

数　字	一元编码	长　度	偏　移	γ 编　码
0	0			
1	10	0		0
2	110	10	0	10,0
3	1110	10	1	10,1
4	11110	110	00	110,00
9	1111111110	1110	001	1110,001
13		1110	101	1110,101
24		11110	1000	11110,1000
511		111111110	11111111	111111110,11111111
1025		11111111110	0000000001	11111111110 ,0000000001

课堂练习

- 计算130的可变字节码
- 计算130的γ编码

Υ编码的长度

- 偏移部分是 $[\log_2 G]$ 比特位
- 长度部分需要 $[\log_2 G] + 1$ 比特位
- 因此，全部编码需要$2[\log_2 G] + 1$比特位
- Υ 编码的长度均是奇数
- Υ 编码在最优编码长度的2倍左右
  - 假定间隔G的出现频率正比于$\log_2 G$—实际中并非如此)
  - (assuming the frequency of a gap G is proportional to log2 G – not really true)

γ编码的性质

- γ编码是前缀无关的，也就是说一个合法的γ编码不会是任何一个其他的合法γ编码的前缀，也保证了解码的唯一性。
- 编码在最优编码的2或3倍之内
- 上述结果并不依赖于间隔的分布！
- 因此，γ编码适用于任何分布，也就说γ编码是通用性(universal)编码
- γ编码是无参数编码，不需要通过拟合得到参数

γ 编码的对齐问题

- 机器通常有字边界 – 8, 16, 32 位
- 按照位进行压缩或其他处理可能会较慢
- 可变字节码通常按字边界对齐，因此可能效率更高
- 除去效率高之外，可变字节码虽然额外增加了一点点开销，但是在概念上也要简单很多

Reuters RCV1索引压缩总表

数据结构	压缩后的空间大小（单位：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（20位）	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

- 现在我们可以构建一个空间上非常节省的支持高效布尔检索的索引
- 大小仅为文档集中文本量的10-15%
- 然而，这里我们没有考虑词项的出现位置和频率信息
- 因此，实际当中并不能达到如此高的压缩比

Q6：压缩后的倒排索引有多大？

Google索引的大小

- 倒排记录的大小(词项-文档对):
  - 文档数目: ~10^{10}
- 每个文档中的独立词项数目： ~250
  - 每个文档词条数为500
- 每个倒排项doc-id平均长度6位
- 倒排表大小： 10^{10} x 250 x 6位 = 1.9 TB
  - 减小到未压缩前的15%（ 12.5 TB）

- 可以存储在一块磁盘上 :-)

Q7: 使用压缩后的倒排索引查询“information retrieval”需要多长时间？

- 假设两个词项的选择率：
  - Information在10亿（10^9）个网页中出现
  - retrieval 在1千万（10^7）个网页中出现

查询处理流程

- 倒排记录表大小（每个doc-id 6位）：
  - Information：10^9 * 6 bits = 750MB
  - Retrieval：10^7 * 6 bits = 7.5 MB
- 磁盘传输时间：
  - 7.5秒（information）+ 0.08秒（ retrieval）
- 忽略CPU耗时和磁盘延迟

查询处理流程

- 查询响应时间由3天降低到50.5秒！
- 继续降低到7.58秒
- :-)

- 但是... 仍然太慢...
- :-(

提前终止查询（1）

- 假设对倒排表的doc-id进行重新排序，使得最相关的文档id排在前面
  - 例如，按tf.idf值，对retrieval词项对应的文档id进行排序
  - [<retrieval; 7; [98, 23, 180, 81, 98, 121, 2, 126,]>
  - 对应查询retrieval可快速返回前十个相关文档：从倒排记录表中读取前10个doc-id后即可提前结束
  - 该方法的问题：
    - 不能支持倒排表压缩、倒排记录表的合并

提前终止查询（2）

- 对所有文档定义一个静态（或全局的）打分
  - 如Google PageRank
  - 按PageRank升序，对doc-id进行重新编号
  - 对应每个词项，PageRank值高的文档将出现在倒排记录表的前面部分
  - 估算查询词项的选择率，仅处理部分倒排记录表

(参考Croft, Metzler & Strohman 2009)

Q8：采用提前终止策略后查询时间是多少？

提前终止查询（3）

- 一个文档中包含查询词项的概率
  - information： $10^9 / 10^{10} = 0.1$
  - retrieval： $10^7 / 10^{10} = 0.001$
- 假设词项之间相对独立：
  - 0.1 x 0.001 = 0.0001的文档同时包含两个词项
  - 即平均每1 / 0.0001 = 10,000个文档中有1个文档同时包含 information和retrieval
- 返回前300个结果，需要处理3,000,000个文档
- 占总倒排记录的比例：$3{,}000{,}000 / 10^{10} = 0.0003$

提前终止查询（3）

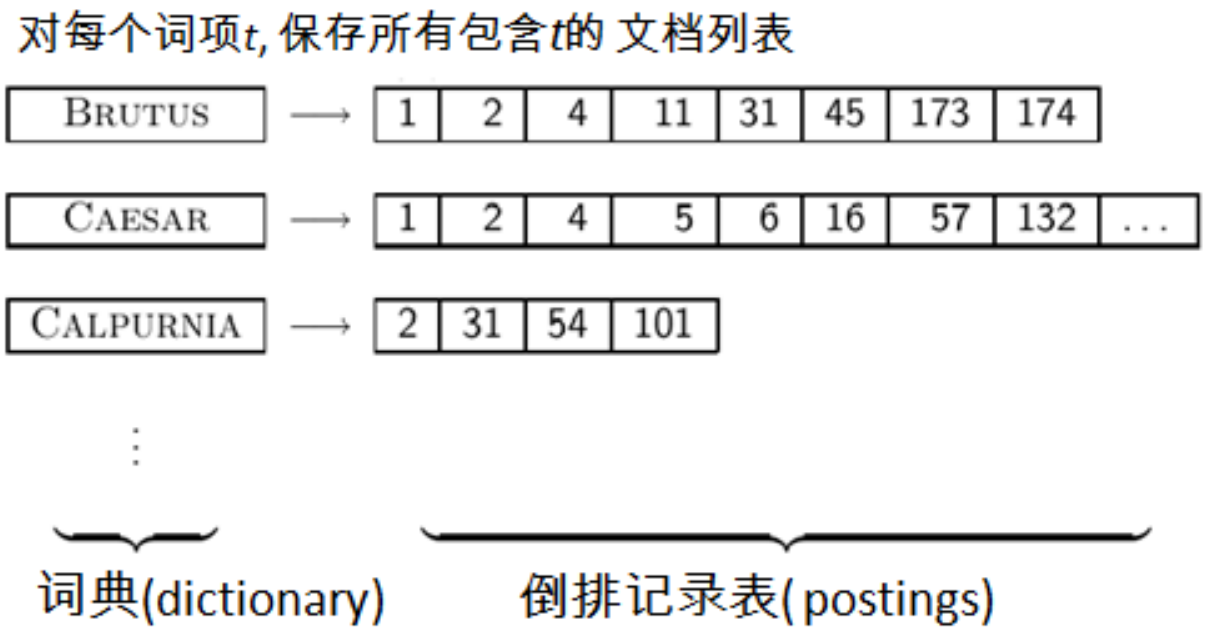
- 因此，只需处理0.0003的倒排记录
  - 0.0003 *750 Mb = 225 kb for information
  - 0.0003 *7.5 Mb = 2.25 kb for retrieval
- 磁盘传输时间：
  - 2 毫秒(information) + 0.02毫秒(retrieval)

查询处理流程

- 查询响应时间由3天降低到2毫秒！！！
- :-)

- *“This engine is incredibly, amazingly, ridiculously fast!”*
  - (from “Top Gear”)

本讲小结



- 信息检索中进行压缩的动机
- 倒排索引中词典部分如何压缩？长字符串方法及改进
- 倒排索引中倒排记录表部分如何压缩？可变字节码、 γ编码
- 词项统计量: 词项在整个文档集中如何分布？ 两个定律

参考资料

- 《信息检索导论》第5章
- 有关字对齐二元编码的原文Anh and Moffat (2005); 及 Anh and Moffat (2006a)
- 有关可变字节码的原文Scholer, Williams, Yiannis and Zobel (2002)
- 更多的有关压缩 (包括位置和频率信息的压缩)的细节参考 Zobel and Moffat (2006)

参考资料(续)

- Alistair Moffat：http://people.eng.unimelb.edu.au/ammoffat/
- Torsten Suel：http://cis.poly.edu/suel

Alistair Moffat

Torsten Suel

参考资料(续)

Jeff Dean's WSDM 2009 keynote:

Challenges in Building Large-Scale Information Retrieval Systems

http://research.google.com/people/jeff/WSDM09-keynote.pdf

http://videolectures.net/wsdm09_dean_cblirs/

课后练习

- 有待补充