

# 信息组织与检索

## 第5讲：索引压缩

主讲人：张蓉

华东师范大学 数据科学与工程学院

# 提纲

---

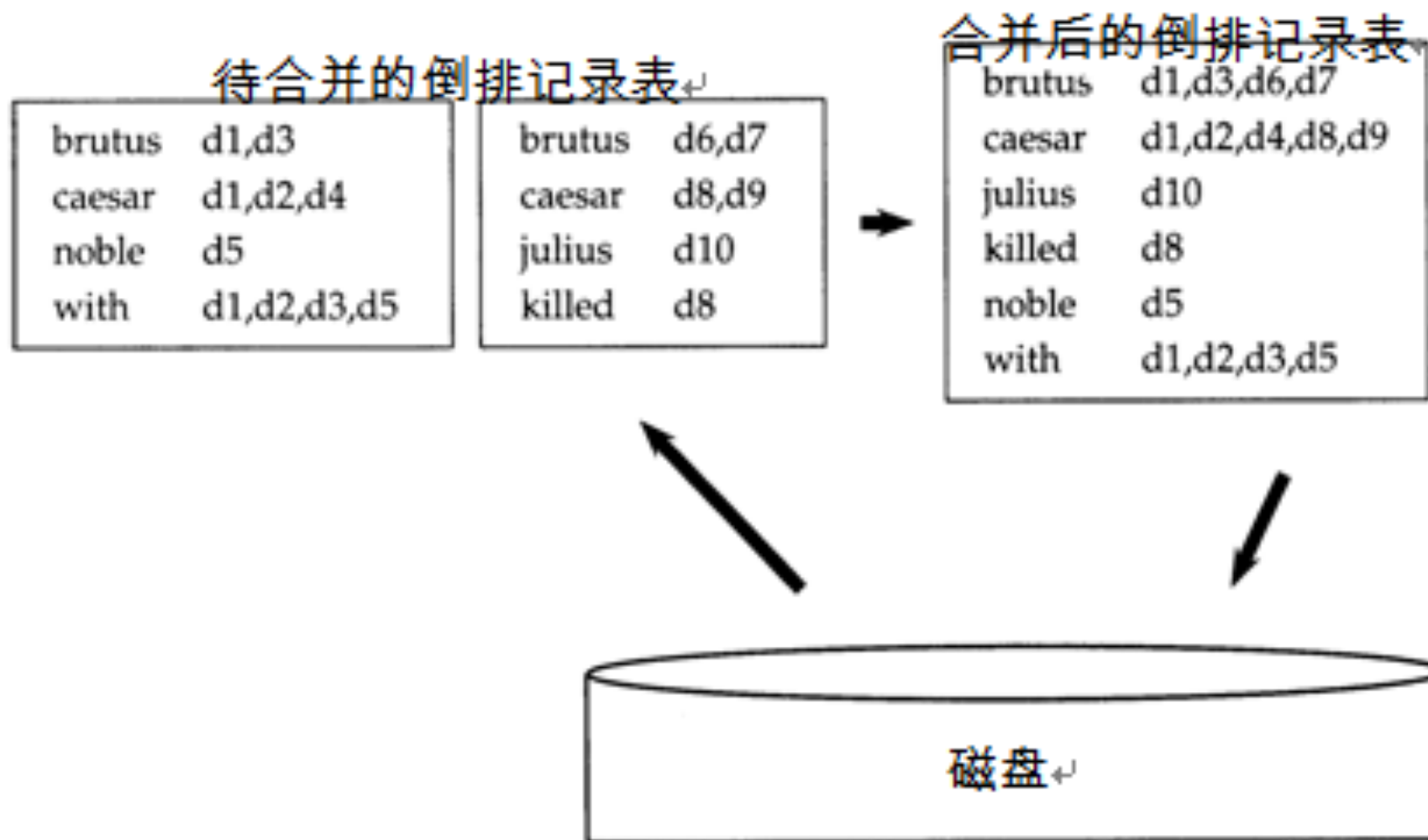
- 上一讲回顾
- 压缩
- 词项统计量
- 词典压缩
- 倒排记录表压缩

# 提纲

---

- 上一讲回顾
- 压缩
- 词项统计量
- 词典压缩
- 倒排记录表压缩

# 基于块的排序索引构建算法BSBI

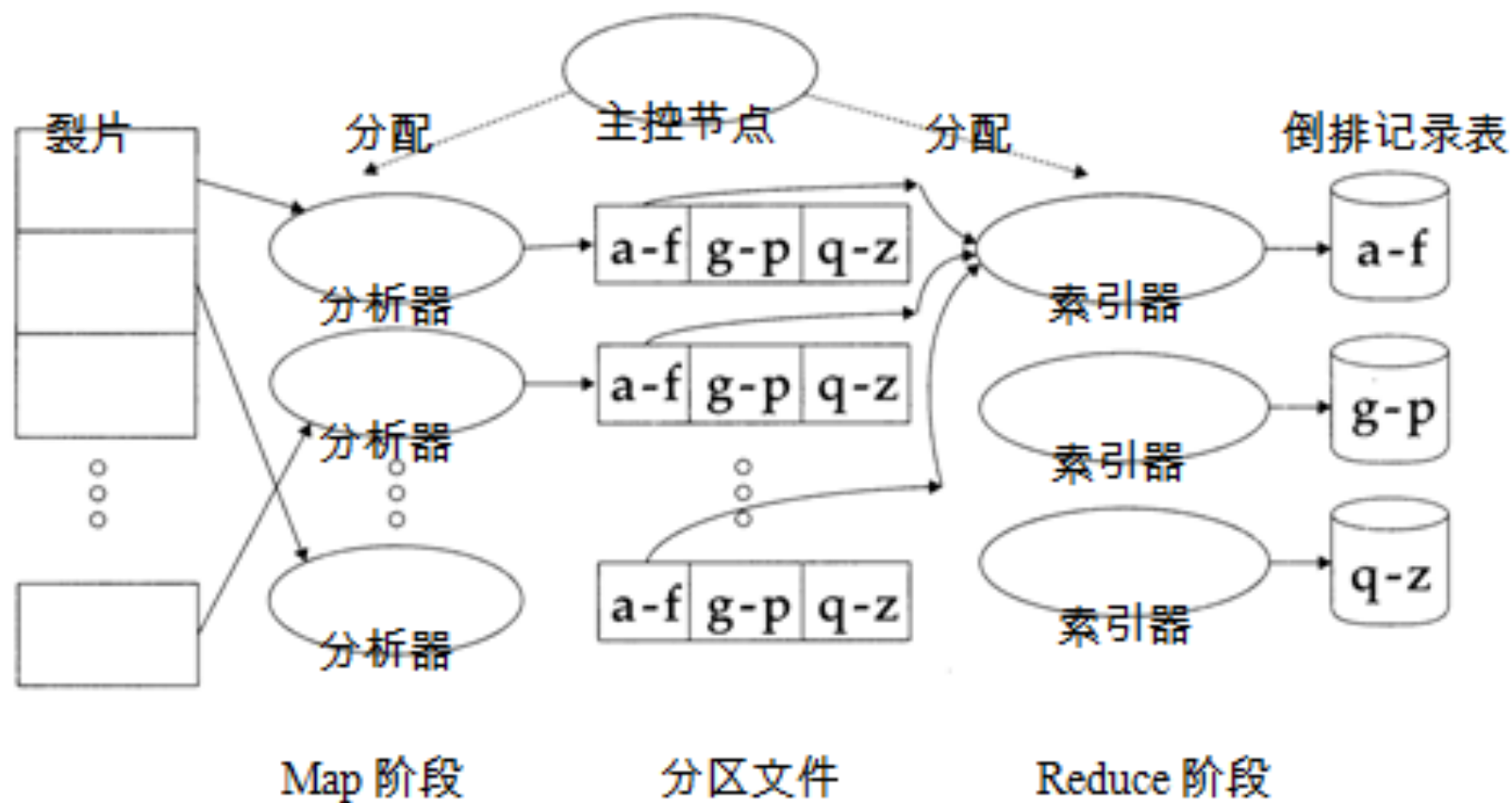


# 内存式单遍扫描索引构建算法SPIMI

---

- **关键思想 1:** 对每个块都产生一个独立的词典 – 不需要在块之间进行term-termID的映射
- **关键思想2:** 对倒排记录表不需要排序，按照他们出现的先后顺序排列即可
- 基础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一个大索引

# 基于MapReduce的索引构建



# 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

# 本讲内容

---

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

对每个词项 $t$ , 保存所有包含 $t$ 的文档列表

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)



# 提纲

---

- 上一讲回顾
- 压缩
- 词项统计量
- 词典压缩
- 倒排记录表压缩

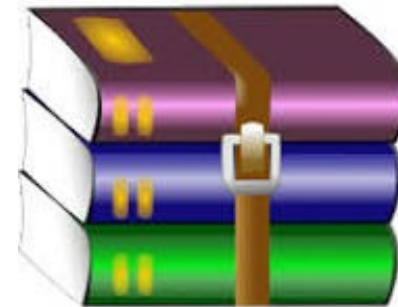
# 什么是压缩?

---

- 将长编码串用短编码串来代替
  - 11111111111111111111 → 18个1



winrar



# 为什么要压缩? (通常来说)

---

- 减少磁盘空间 (节省开销)
- 增加内存存储内容 (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, 例子?
- 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, 例子?
  - 索引压缩中通常都使用无损压缩



# 提纲

---

- 上一讲回顾
- 压缩
- 词项统计量
- 词典压缩
- 倒排记录表压缩

# 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



# 预处理的效应(有损)

	不同词项			无位置信息倒排记录			词 条 <sup>①</sup>		
	数目	$\Delta\%$	T%	数目	$\Delta\%$	T%	数目	$\Delta\%$	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

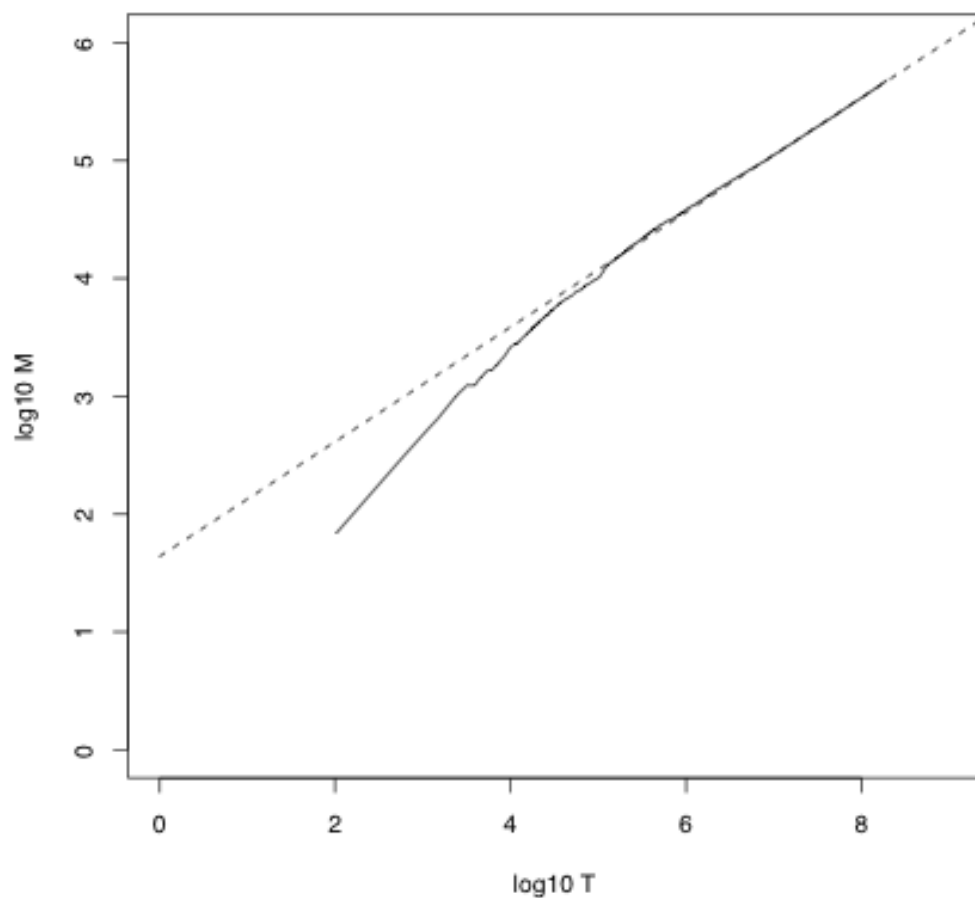
# 第一个问题：词汇表有多大

(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

## 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 a/an/the, 的 了 地 我

- Zipf定律: 第 $i$ 常见的词项的频率 $cf_i$ 和 $1/i$ 成正比

$$cf_i \propto \frac{1}{i}$$

- $cf_i$  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定律: 第 $i$ 常见的词项的频率 $cf_i$ 和 $1/i$ 成正比

$$cf_i \propto \frac{1}{i}$$

$cf_i$  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*)出现次数为  $cf_2 = \frac{1}{2}cf_1 \dots$

- ... 第三常见的词项 (*and*) 出现次数为  $cf_3 = \frac{1}{3}cf_1$

- 另一种表示方式:  $cf_i = c \cdot i^{-k}$  或  $\log cf_i = \log c + k \log i$  ( $k = -1$ )

- 幂定律(power law)的一个实例

- 生活中的zipf定律

- 旅游城市的游客访问数量

# 生活中的zipf定律

---

- 80%的财富集中在20%的人手中
- 80%的用户只使用20%的功能
- 20%的用户贡献了80%的访问量
- 完美的二八定律



# 生活中的zipf定律

---

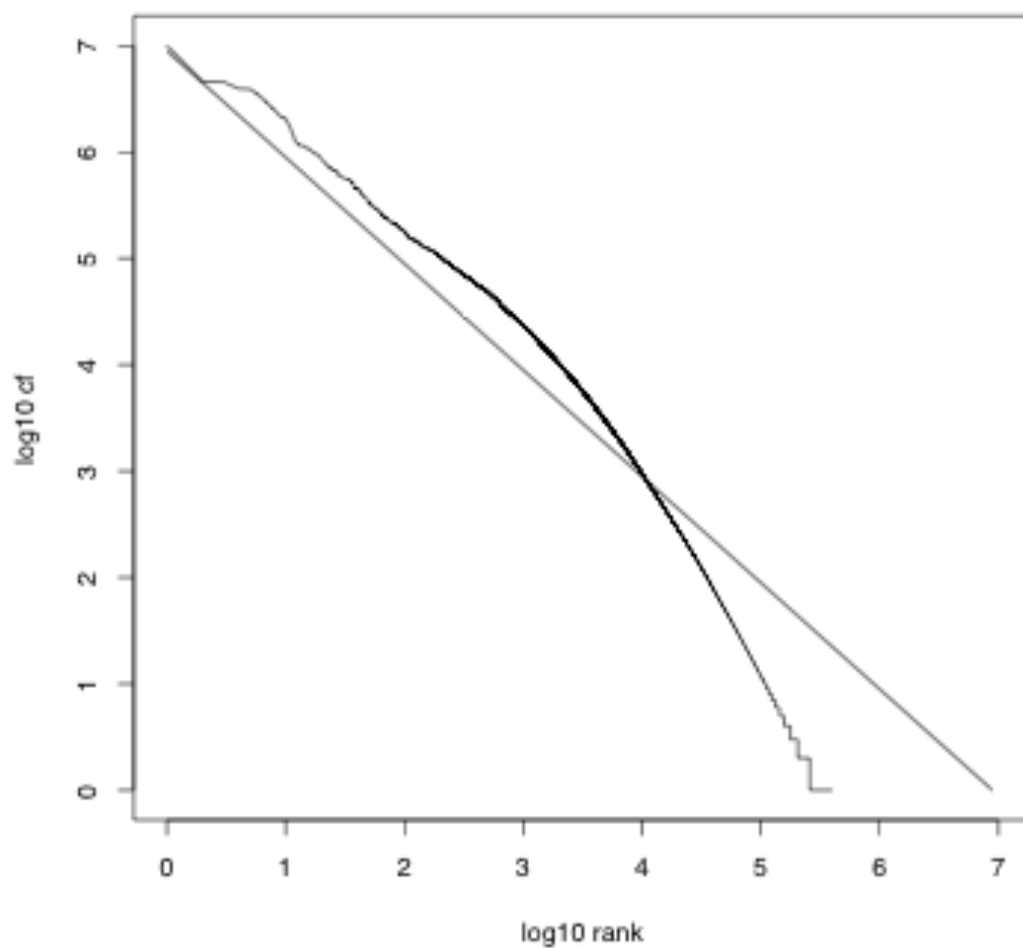
## ■ 网店

- 网店相当于用极小的成本，便换来几乎无限长的货架
- 可以同时包括大热门以及几乎无人问津的众多冷门

## ■ 能否想几个例子？



# 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

---

- 上一讲回顾
- 压缩
- 词项统计量
- 词典压缩
- 倒排记录表压缩

# 词典压缩

- 相对于倒排记录表，词项较小
- 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	→

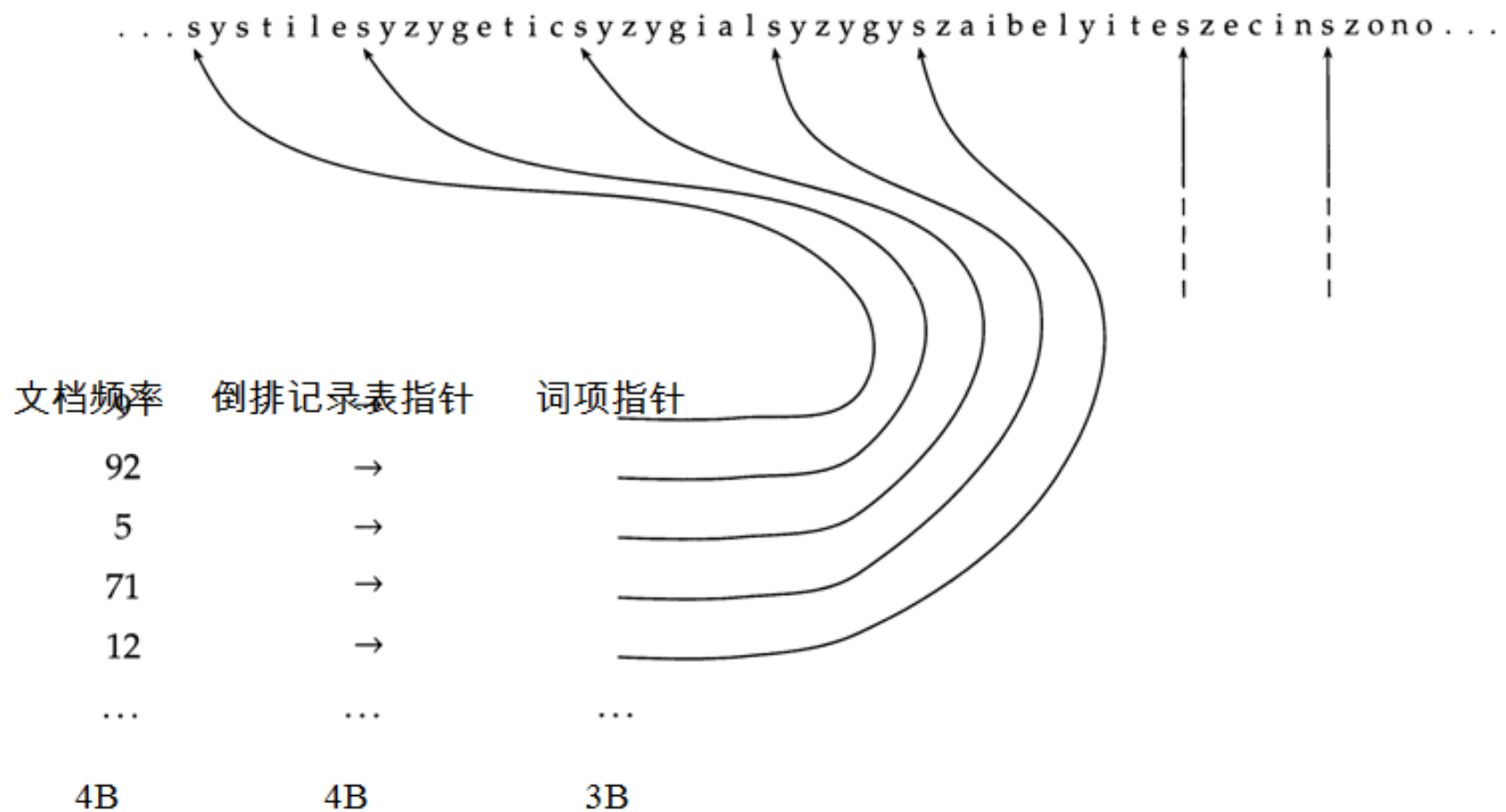
空间需求:      20 字节      4 字节      4 字节

对Reuters RCV1语料:  $(20+4+4)*400,000 = 11.2 \text{ MB}$

# 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
    - a an
- 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
- **提问** 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 ( $8 \times 400000$  个位置需要  $\log_2(8 \times 400000) < 24$  位来表示)
- 空间消耗:
  - $400,000 \times (4 + 4 + 3 + 8) = 7.6\text{MB}$  (而定长数组方式需要11.2MB)

# 单一字符串方式下按块存储



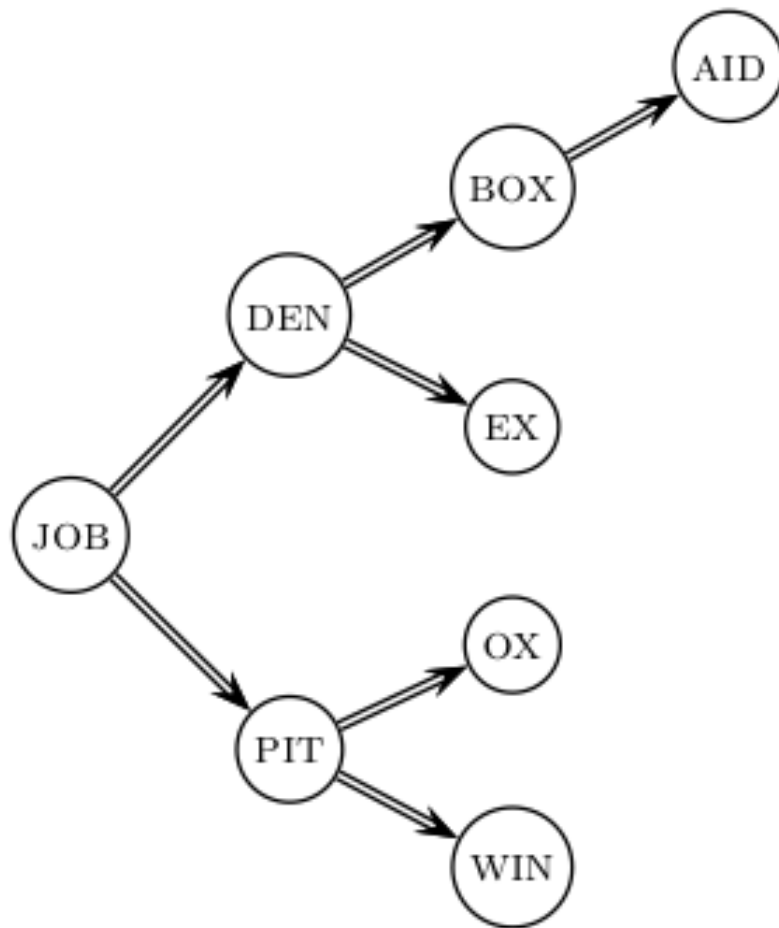


# 按块存储下的空间消耗

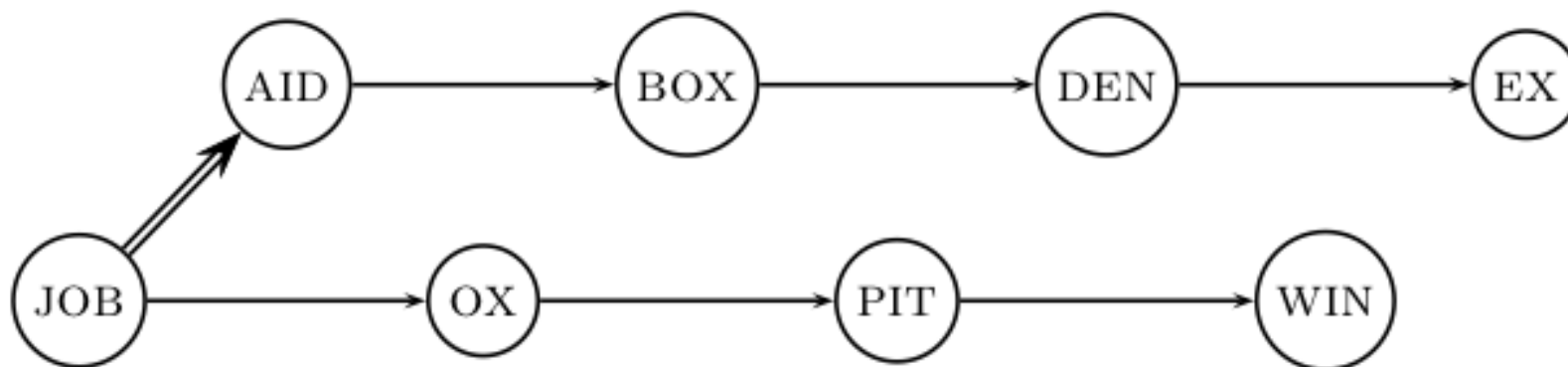
---

- 如果不按块存储，则每4个词项指针将占据空间  $4 \times 3 = 12\text{B}$
- 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  $3+4=7\text{B}$
- 因此，每4个词项将节省  $12-7=5\text{B}$
- 于是，整个词典空间将节省  $40,000/4 \times 5\text{B} = 0.5\text{MB}$
- 最终的词典空间将从  $7.6\text{MB}$  压缩至  $7.1\text{MB}$

# 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 ...

8 a u t o m a t a 8 a u t o m a t e 9 a u t o m a t i c 10 a u t  
o m a t i o n



- ... 可以采用前端编码方式继续压缩
- 8 a u t o m a t \* a 1 ◊ e 2 ◊ i c 3 ◊ i o n

1/2/3表示什么?

表示词项长度, 比如1表示 e只有一个字符

# 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

# 提纲

---

- 上一讲回顾
- 压缩
- 词项统计量
- 词典压缩
- 倒排记录表压缩

# 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

怎么做？

# 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:  $283159 - 283154 = 5$ ,  $283202 - 283154 = 43$
- 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	2	...
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			

存在什么问题？

罕见词的处理和常见词，间隔大小差别太大

# 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
- 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比特, 那么 *c* 置 1, 将间隔编入一个字节的后7位中
- 否则: 将前面字节的延续位置0, 其他7位放入当前字节中, 最后一个字节的延续位置1 (表示结束)

# 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

$$824=1100111000$$

# VB 编码算法

VBENCODENUMBER( $n$ )

```
1  $bytes \leftarrow \langle \rangle$ 
2 while  $true$ 
3   do PREPEND( $bytes, n \bmod 128$ )
4     if  $n < 128$ 
5       then BREAK
6      $n \leftarrow n \text{ div } 128$ 
7    $bytes[\text{LENGTH}(bytes)] += 128$ 
8 return  $bytes$ 
```

VBENCODE( $numbers$ )

```
1  $bytestream \leftarrow \langle \rangle$ 
2 for each  $n \in numbers$ 
3   do  $bytes \leftarrow \text{VBENCODENUMBER}(n)$ 
4      $bytestream \leftarrow \text{EXTEND}(bytestream, bytes)$ 
5 return  $bytestream$ 
```

# VB编码的解码算法

VBDECODE(*bytestream*)

```
1  numbers  $\leftarrow \langle \rangle$ 
2  n  $\leftarrow 0$ 
3  for i  $\leftarrow 1$  to LENGTH(bytestream)
4  do if bytestream[i] < 128
5      then n  $\leftarrow 128 \times n + \textit{bytestream}[i]$ 
6      else n  $\leftarrow 128 \times n + (\textit{bytestream}[i] - 128)$ 
7          APPEND(numbers, n)
8          n  $\leftarrow 0$ 
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

# Y编码

- 另外一种变长编码是基于位的编码
- $\gamma$  编码是其中最出名的一种
- 首先，在介绍 $\gamma$ 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

110

- 70的一元码是：

[illegible]



# Y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  $13 \rightarrow 1101 \rightarrow 101 = \text{偏移}$
- 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Y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，即1110 101

# Y编码的例子

数 字	一元编码	长 度	偏 移	$\gamma$ 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	111111110	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 $\gamma$ 编码

# γ编码的长度

---

- 偏移部分是  $\lfloor \log_2 G \rfloor$  比特位
- 长度部分需要  $\lfloor \log_2 G \rfloor + 1$  比特位
- 因此，全部编码需要  $2\lfloor \log_2 G \rfloor + 1$  比特位
- γ 编码的长度均是奇数
- γ 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  $\log_2 G$  — not really true)

# $\Upsilon$ 编码的性质

- $\Upsilon$  编码是前缀无关的，也就是说一个合法的  $\Upsilon$  编码不会是任何一个其他的合法  $\Upsilon$  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，  $\Upsilon$  编码适用于任何分布，也就说  $\Upsilon$  编码是通用性 (universal) 编码
- $\Upsilon$  编码是无参数编码，不需要通过拟合得到参数

# 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， $\gamma$ 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