第五章　检索效率优化

小组成员：徐才舒（2120101775）
杨博斐（2120101778）
吴　玥（2120101774）
谷　雨（2120101735）

http://hi.baidu.com/drkevinzhang/

LOGO

本章讲解内容

5.1 倒排索引

1. 构建倒排索引
2. 压缩倒排索引
3. 变长索引压缩
4. 基于倒排表大小的变长压缩
5. 索引剪枝
6. 在构建索引前对文档重新排序

5.2 查询处理

1. 倒排索引的修订
2. 部分结果集检索
3. 简化向量空间

本节讲解内容——徐才舒

5.1 倒排索引

1. 构建倒排索引
2. 压缩倒排索引

查询算法的性能

- 算法的查询处理效果指标：

$$召回率 = \frac{系统检索到的相关文件数}{相关文件总数}$$ 查全率

$$准确率 = \frac{系统检索到的相关文件数}{系统返回的文件总数}$$ 查准率

- 大部分信息检索研究，包括信息检索模型以及实用策略，都集中在如何提高从文档集中找出查询相关文档的准确率和召回率上，因为人们寄希望于计算机硬件速度将会持续不断地增长。但是，实用系统的运行性能也是用户很关注的。

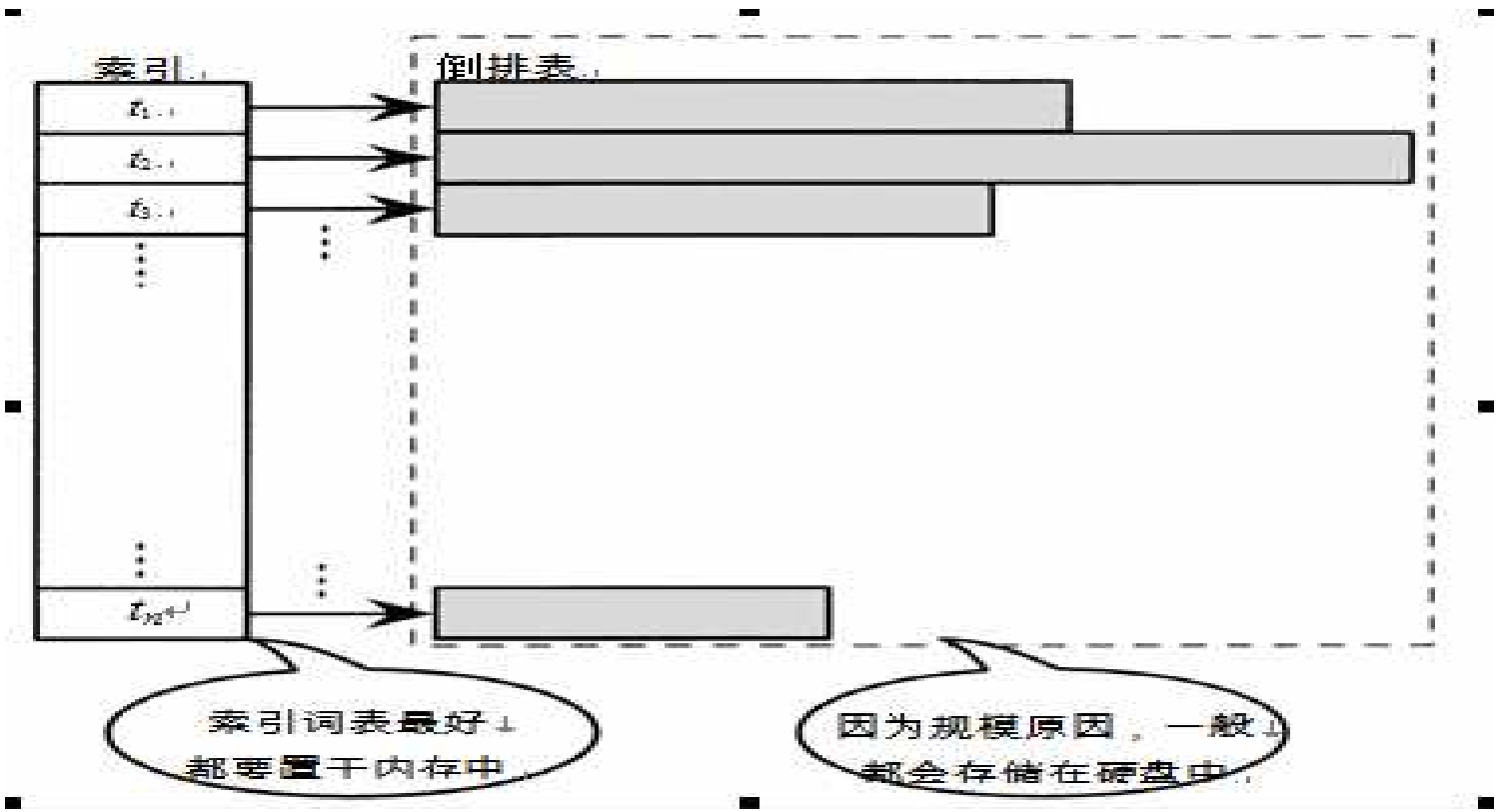
$O(q(tf_{max}))$

查询算法的性能

- 算法的查询时间效率指标：
  检索算法的时间复杂度——
  大多数算法的时间复杂度为 $O(q(tf_{max}))$
  q是查询语句中词的个数，
  tf_{max} 是查询词选择范围的最大值。

  - 本章讲解如何改善串行信息检索算法的性能
  - 第7章将介绍提高并行信息检索算法性能的一种方法——利用多处理器系统
  - 本章所介绍的大多数算法都是直接应用于向量空间模型的，当然这些算法也同样适用于概率模型

- 什么是倒排索引？



$O(q(tf_{max}))$

5.1 倒排索引

- 倒排索引由两部分组成

  ——所有独立的词列表称为索引

  ——词对应的一系列表统称为倒排表

- 索引：对文档进行预处理后，识别出文档集中的每一个词，这些词统称为索引（例如按照字母排序）。
- 倒排表：存储记录哪些文档包含这个索引词。

  ——每个条目可以包含词在文档中的位置信息（如词位置、句子、段落），有利于实现邻近搜索；

  ——每个条目可以包含词在该文档中的权重值，用于文档的相关性计算，根据相关度对所有文档排序，然后识别出所有与给定查询相关的文档，呈现给用户。

$\Phi(q(tf_{max}))$

1 构建倒排索引

- 早期的信息检索系统只处理非常小的文档集，之后随着文档集规模的不断增大，人们引入了倒排索引来提高查询的处理性能。
- 倒排表：对于集合中的每个词，都有一个包含多元组的倒排表与之关联。为了对文档进行排序，我们还需要保留词频和权重信息。
- 多元组的倒排表：集合每个词的元组均采用<doc_id,tf>的形式来表示。

- 倒排索引实例：
- 假设文档集由文档1和文档2共计两篇文档构成，文档1中词“sales”出现两次，“vehicle”出现一次，文档2中“vehicle”出现一次。那么，索引将包含两个条目：“sales”和“vehicle”。倒排表仅仅是一个与每个索引词相关联的链接表。在本例中，倒排表如下所示：
- sales→(1,2)
- vehicle→(1,1)(2,1)

- 构建倒排索引的一种可行算法如下：

  按顺序扫描整个文档集，一次处理一个词。该算法的输出结果是一些存储在磁盘上的文件。有如下几种文件：

- 索引文件：该文件包含了集合中每个词的倒排表。如果词*t*在*n*个不同的文档中均出现了，*t*的倒排表如下所示：

$$t \rightarrow (d_1, tf_{1,j}), (d_2, tf_{2,j}), \mathrm{L}\ , (d_i, tf_{i,j})$$

  其中*di*表示文档*i*的标识符，$tf_{i,j}$表示词*j*在文档*i*中的出现次数。

- 文档文件：该文件含有每篇文档的各种信息，比如文档ID、文档名、发表日期等。

- 权重文件：该文件存储的是文档集中每篇文档的权重。这个权重是余弦相似度计算中的分母，余弦相似度指的是查询向量与文档向量夹角的余弦值。

1 构建倒排索引

词项的权重值：

- *t* —— 文档集中不同词项的个数。
- *tfij* —— 词项*tj*在文档*Di*中出现的次数，也就是词频。
- *dfj* —— 包含词项*tj*的文档的篇数。
- *idfj* —— $\lg\left(\frac{d}{df_j}\right)$ 其中*d*表示所有文档的篇数。这就是逆文档频率。

- 对于每一篇文档向量，都有*n*个分量，每个分量为在整个文档集中计算出来的每个词项的权重。对于整个文档集中每个不同的词项，都包含一个词条。
- 词项在一篇文档中出现的频率越高，则权重越大；相反，如果词项在所有文档中出现的频率越高，则权重越小。
- 对于文档中词项的权重因素，主要综合考虑词频和逆文档频率。也就是说，我们使用下面的公式计算文档*i*对应的向量中第*j*个词条的值：
- *dij = tfij * idfj*

词项的权重值：

- 当我们在一篇文档检索系统中用文档集中t个不同的词项来查询时，系统将为每个文档计算维度为t的向量D(di1, di2,…,dit)。向量值使用前文所述的词项权重填充。类似地，查询中的词项构建的向量为Q(wq1, wq2,…,wqt)。
- 查询Q和文档Di的相似度可以简单地定义为两个向量的内积。因为查询向量和和文档向量在长度上是相似的，这种策略也常常被用来计算两篇文档的相似度。

$$SC(Q,D_i)=\sum_{j=1}^{t} w_{qj} \times d_{ij}$$

构建倒排索引

构建倒排索引的具体流程：

- 扫描整个文档集，每次只处理一个词
- 检查该词是否是一个停用词或是否已经判别过
- 用一个散列函数来快速定位该词在数组中的位置
- 检查倒排表中第一个条目的文档ID是否与待处理词的文档ID一致

——如果一致，则将倒排表中第一个条目的词频加1

——如果不一致，则需要在倒排表的开始处添加一个条目

构建倒排索引

构建倒排索引的具体流程：

- 每当新加入一个倒排表条目时，都需要为其动态分配内存。检查为构建索引所预留的内存是否已经用完——如果用完，则暂停处理，将内存中的倒排表全部写到磁盘，然后再继续处理。向磁盘写入的倒排表中，每个词的倒排表都是连续存储的。
- 当所有词都处理完毕后，索引就构建完成了。
- 扫描每个词的倒排表来计算其逆文档频率。
- 计算完逆文档频率，接着计算文档权重。

- **倒排索引的作用：**倒排表中的条目按照文档标识符升序排列，构建倒排索引的成本较高，但是一旦构建完成，便能有效地进行查询，能大大减少ad hoc查询时所需的I/O量。
- **倒排索引的代价：**构建时需要生成三种文件，存储索引需要空间；构建时扫描整个文档集，存储到辅助存储器上需要大量的I/O操作，构建索引需要时间。
- **倒排索引的使用步骤：**每接收到一个查询——先查询索引——检索到相应的倒排表——最后检索算法根据倒排表的内容对文档进行排序。

$\Phi_{rsv}(q(tf_{max}))$

1 构建倒排索引

倒排索引的几点问题：

- 索引文件的大小决定其所占用的存储开销。
- 未压缩的索引通常需要4个字节来存储文档ID；长文档中有些词的出现次数有可能超过255次，需要采用2个字节来存储词频。
- 大多数索引的大小和原始文档差不多，这意味着所需要的存储空间将是原始文档大小的两倍。
- 大索引文件需要更大的I/O带宽来读取，其大小也直接影响了处理时间。
- 将高效的倒排索引存储到辅助存储器上也需要大量的I/O操作。
- 文档权重的计算也与逆文档频率一样，需扫描每个词的倒排表。

我们知道，倒排索引文件研究的一个关键目标就是研究一些算法来减少I/O带宽和存储开销。因此我们进入下一部分的讲解。

2 压缩倒排索引

- 在Moffat和Zobel的研究中，他们使用了一种相对比较容易解压缩的索引生成方式。压缩索引后的大小不到原始文档集的10%，而且这里面还包含了停用词。

- 下面来介绍一下压缩倒排索引的有关内容：

2 压缩倒排索引

- 倒排索引中倒排表的大小可以通过齐普夫分布来估计。
- 它描述了自然语言中每个词的词频分布情况：如果把词出现的频率按由大到小排列，那么每个词的词频与它排序序号的乘积是一个常量。
- 常量取1时齐普夫分布中前5个词的词频分布如下：

排序序号	频率	常量
1	1.00	1
2	0.50	1
3	0.33	1
4	0.25	1
5	0.20	1

- 使用公式（其中*r*是词的排序序号，*C*为常量值），我们可以估计出给定词的出现次数。常量*C*与词所处领域相关，通常等于词频最高词的出现次数。（根据齐普夫定律，大多数词都很少出现，大约有一半的词只出现一次）。

$\Phi_{max}(q(tf_{max}))$

- 哈夫曼编码举例
- 哈夫曼编码——一种一致性编码法（又称“熵编码法”），用于数据的无损耗压缩。

  ——使用一张特殊的编码表将源字符（例如某文件中的一个符号）进行编码。
- 主要思想：这张编码表的特殊之处在于，它是根据每一个源字符出现的估算概率而建立起来的。具体是出现概率高的字符使用较短的编码，反之出现概率低的则使用较长的编码，这便使编码之后的字符串的平均期望长度降低，从而达到无损压缩数据的目的。

- 哈夫曼编码举例
- 在英文中，e的出现概率很高，而z的出现概率则最低。当利用哈夫曼编码对一篇英文进行压缩时，e极有可能用一个位(bit)来表示，而z则可能花去25个位（不是26）。用普通的表示方法时，每个英文字母均占用一个字节（byte），即8个位。二者相比，e使用了一般编码的1/8的长度，z则使用了3倍多。倘若我们能实现对于英文中各个字母出现概率的较准确的估算，就可以大幅度提高无损压缩的比例。

$\Phi_{rsv}(q(tf_{max}))$

2 压缩倒排索引

- 固定长度的索引压缩介绍
- **字节对齐压缩（Byte-Aligned）**
- 字节对齐压缩即对于一个给定的正整数，用一个或多个字节表示。表示该数首字节最左边的两位为长度指示器（length indicator），剩余位可以用来存储实际的数。文档ID不同，记为x，文档ID需要基于x的字节标识码，用前面所说的2bits写下长度指示器。写下x的二进制表示法，如下例：
- 0-63　　00xxxxxx
- 64-(16K-1)　01xxxxxx xxxxxxxx
- 16K-(4M-1)　10xxxxxx xxxxxxxx xxxxxxxx
- 4M-(1G-1)　11xxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
-
- 0　　00000000
- 1　　00000001
- …　　…
- 63　　00111111
- 64　　01000000 01000000
- 65　　01000000 01000001
- 字节对齐压缩的优点是容易编码和解码，位操作少，占用CPU时间少，缺点是对很小的整数压缩率低，每个整数最少用一个字节的空间。
- 通过使用字节边界来实现BA压缩，运行时会付出些许代价，但可以获得一定的压缩比。BA算法易于实现，而且压缩比比较高（使用停用词后，压缩文件大小占未压缩索引文件大小的比例大约是15%）。

$\Phi_{idf}(q(tf_{max}))$

2 压缩倒排索引

- 固定长度的索引压缩介绍
- 倒排表的每个条目是按照文档ID升序排列的。当倒排索引经过删除变动操作后，这种文档的顺序就会出现例外。因此，RLE（Run-Length Encoding，行程长度编码）方法适用于文档ID的编码。
- 对于任意一个文档ID，仅仅需要计算当前ID以及待处理ID之间的偏移量。对于没有其他文档ID存在的情况下，我们就存储文档ID的压缩形式。采用这种技术，相对较小的数值可以保证在整个数据中占较大比重。
- 这种方案可以很有效减小ID的范围，允许它们以更精小的形式存储。随后，采用接下来的方法用于数据压缩。对于一个给定的输入值，保留最左面的两个比特，用来存储整个存储值占用的字节数。两个比特位的表现形式有4种可能的组合，因此，对于所有的文档ID，可以使用一个两比特位的指示器。整型数据的存储长度可以为6比特、14比特、22比特或者30比特。最佳的情况下，所有的值都小于2的6次方＝64，通过上述方法，每个单独的数据记录大小使用一个字节就可以存储。而在没有压缩的情况下，所有的文档ID都需要使用4字节存储。

$\Phi_{max}(q(tf_{max}))$

2 压缩倒排索引

- 固定长度的索引压缩介绍
- 对于待压缩的每一个值，需要计算存储值所需的最小字节数。
- 下表给出了我们可以存储整型数据的值范围及其字节数，同样长度指示器分别为1字节、2字节、3字节以及4字节。如果文档集中的文档数目超过2的30次方，这种方法可以将指示器长度扩展为3个比特，这样表示的范围就达到了2的61次方-1。

长　度	所需字节数
$0 \leqslant r < 64$	1
$64 \leqslant r < 16\ 384$	2
$16\ 384 \leqslant r < 4\ 194\ 304$	3
$4\ 194\ 304 \leqslant r < 1\ 073\ 741\ 824$	4

- 对于词频，因为每一个频率值均独立于先前的词频值，因此我们并不需要使用值间偏移量的概念。但是，我们可以使用相同的编码方式。因为，我们并不期望一篇文档中包含的词超过2的15次方＝32 768次，使用一个或两个字节就可以来存储这个数值，其中只需要1比特作为长度指示器。

固定长度的索引压缩示例

- 给定任意一个索引词*t*1，我们考虑其索引的条目，假定*t*1在文档1、文档3、文档7、文档70和文档250中出现。BA压缩使用最高的两个比特位来表明存储该值需要的字节数。对于前4个值，只需要一个字节就可以表示；对于最后一个值——180，需要两个字节来表示。需要注意的是：我们只需要计算倒排表中每一项的差。最后的差值为：250-70＝180。我们需要计算所有的最终数值。这些数值和压缩后的相应比特字符串见下表。（字节对齐方式压缩）

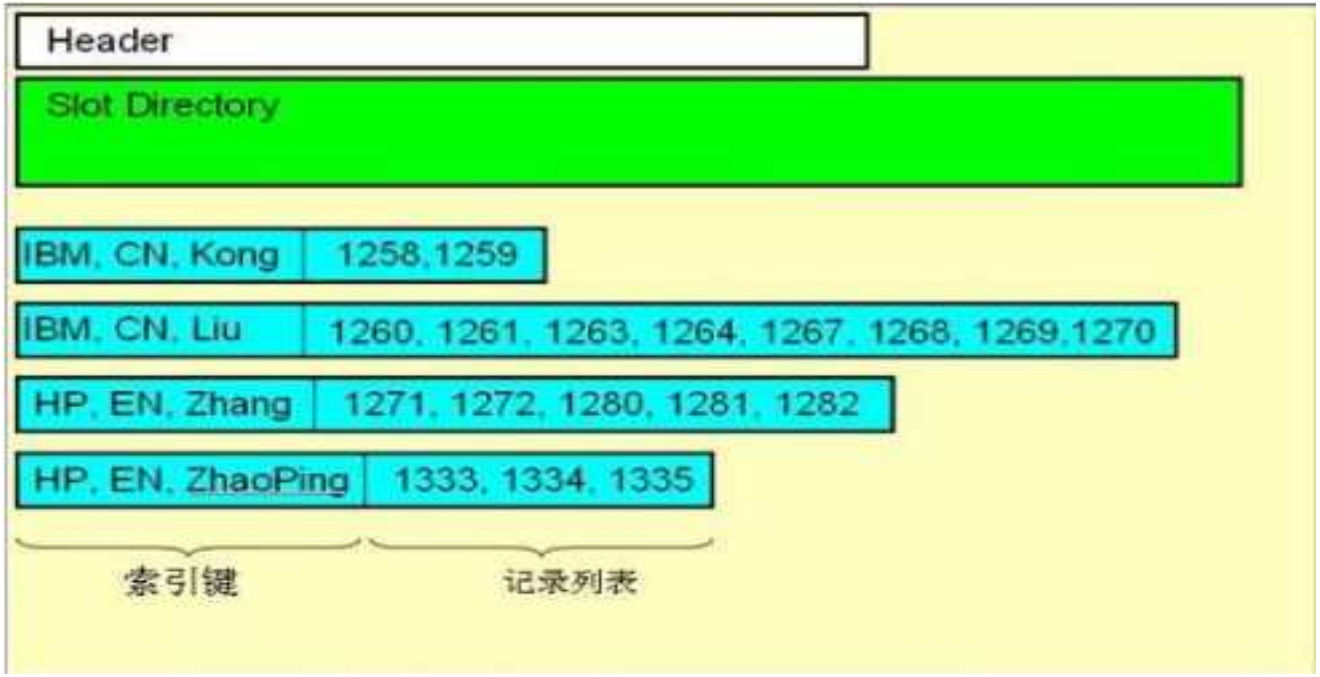
值	压缩的比特串
1	00 000001
2	00 000010
4	00 000100
63	00 111111
180	01 000000 10110100

- 固定长度的索引压缩示例
- 在没有压缩之前，倒排表中的每条记录需要4字节，5条记录总共需要20个字节。这些数值和它们对应压缩前的比特字符串详见下表。（基准线：压缩前）

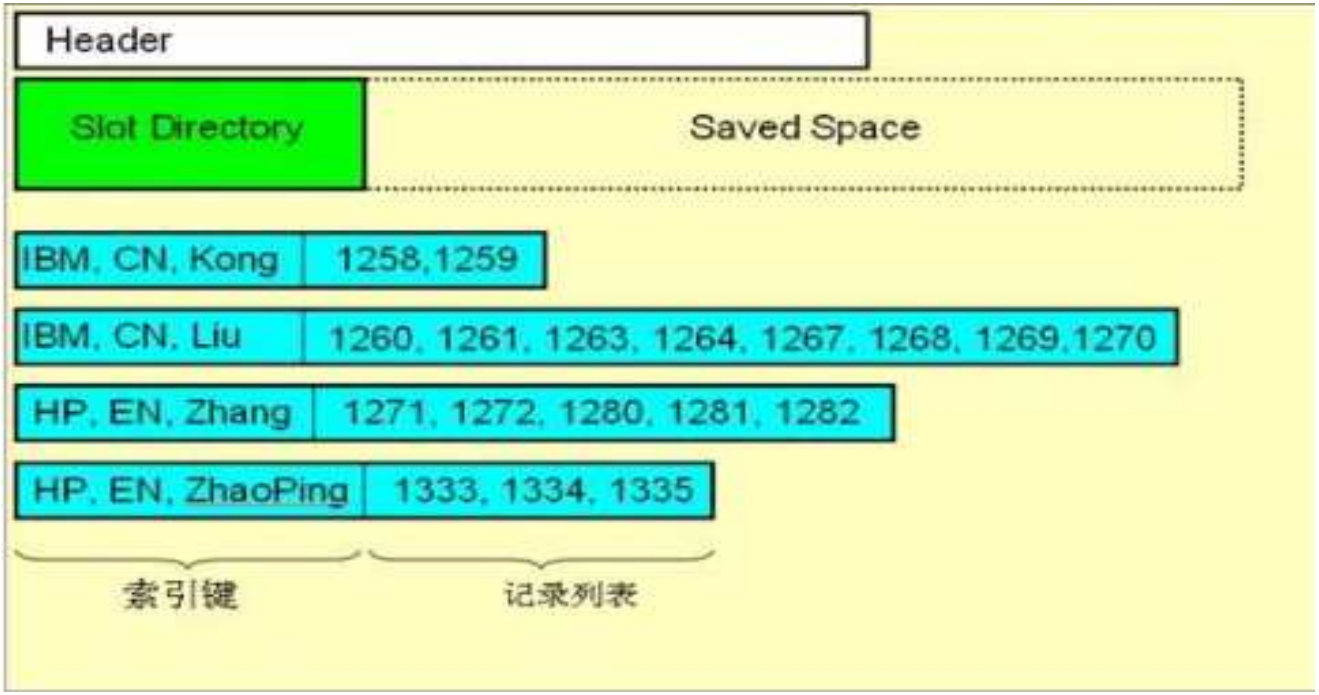
值	未压缩的比特串
1	00000000 00000000 00000000 00000001
3	00000000 00000000 00000000 00000011
7	00000000 00000000 00000000 00000111
70	00000000 00000000 00000000 01000110
250	00000000 00000000 00000000 11111010

- 结论：在这个例子中，未压缩的数据需要160比特，使用BA压缩后仅需要48比特。

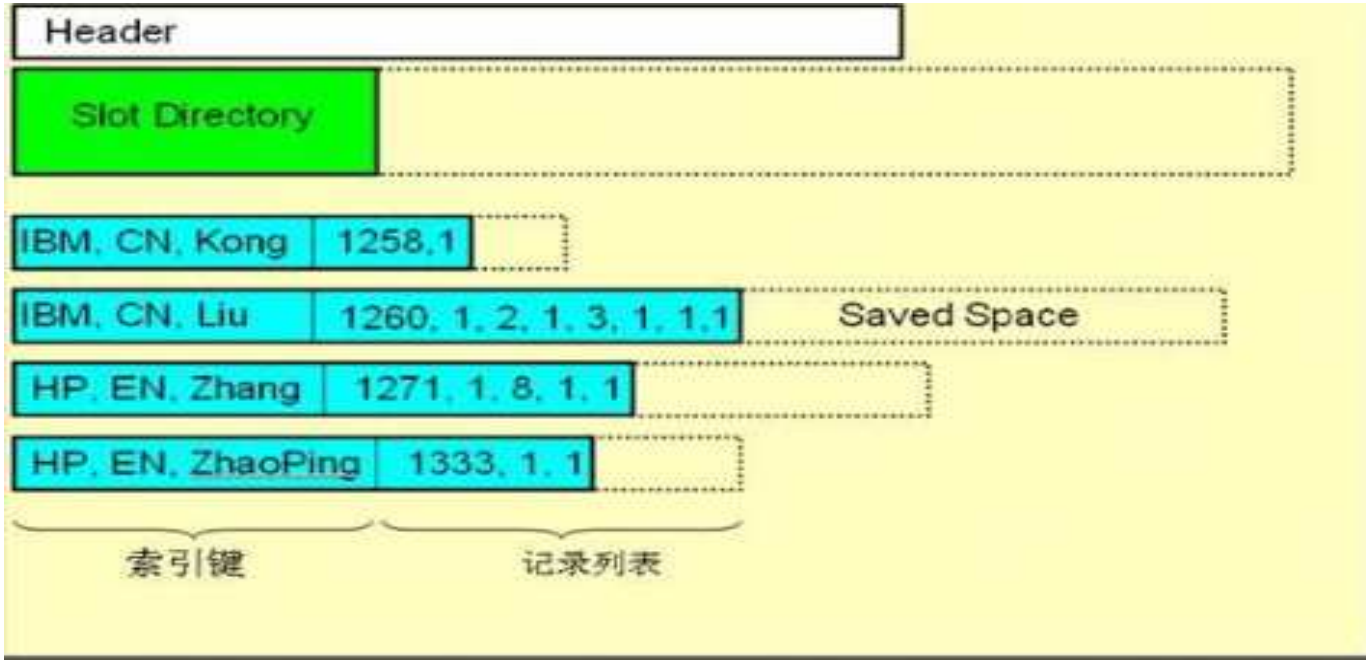
❖ DB2 V9.7索引压缩原理举例



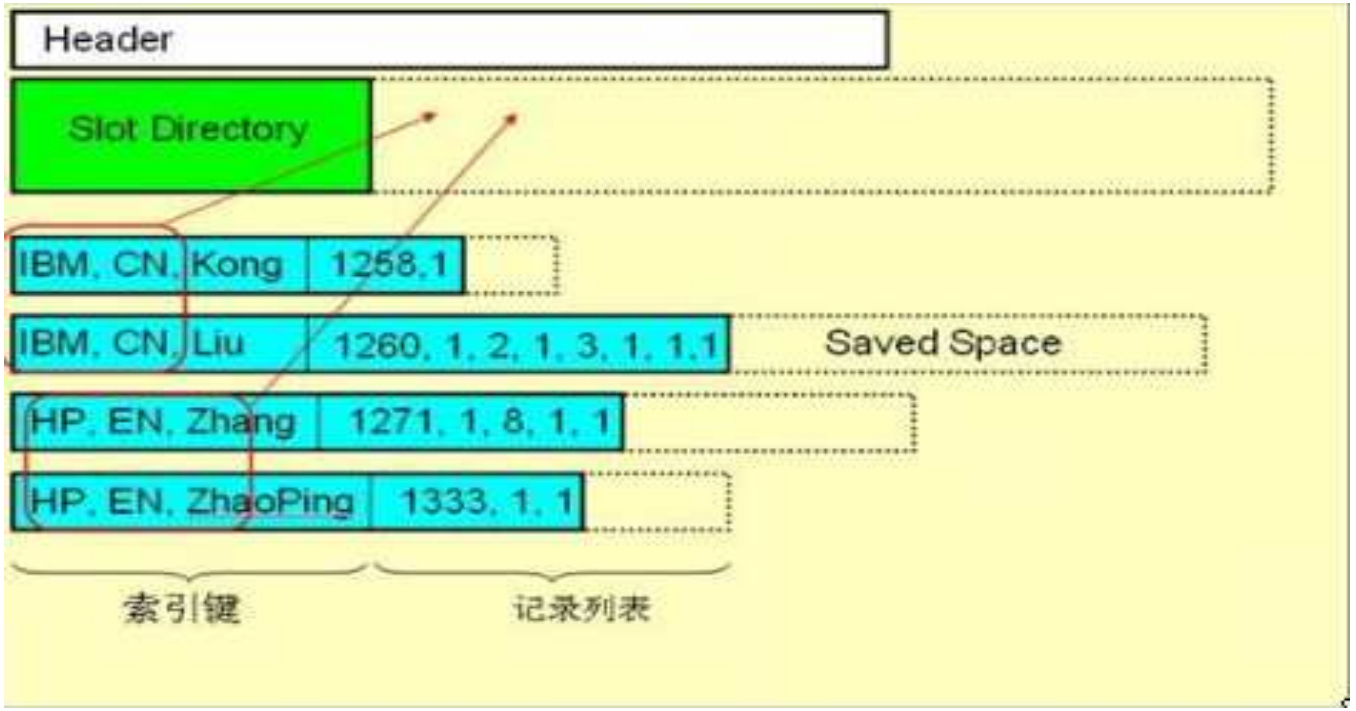
❖ DB2 V9.7索引压缩原理举例



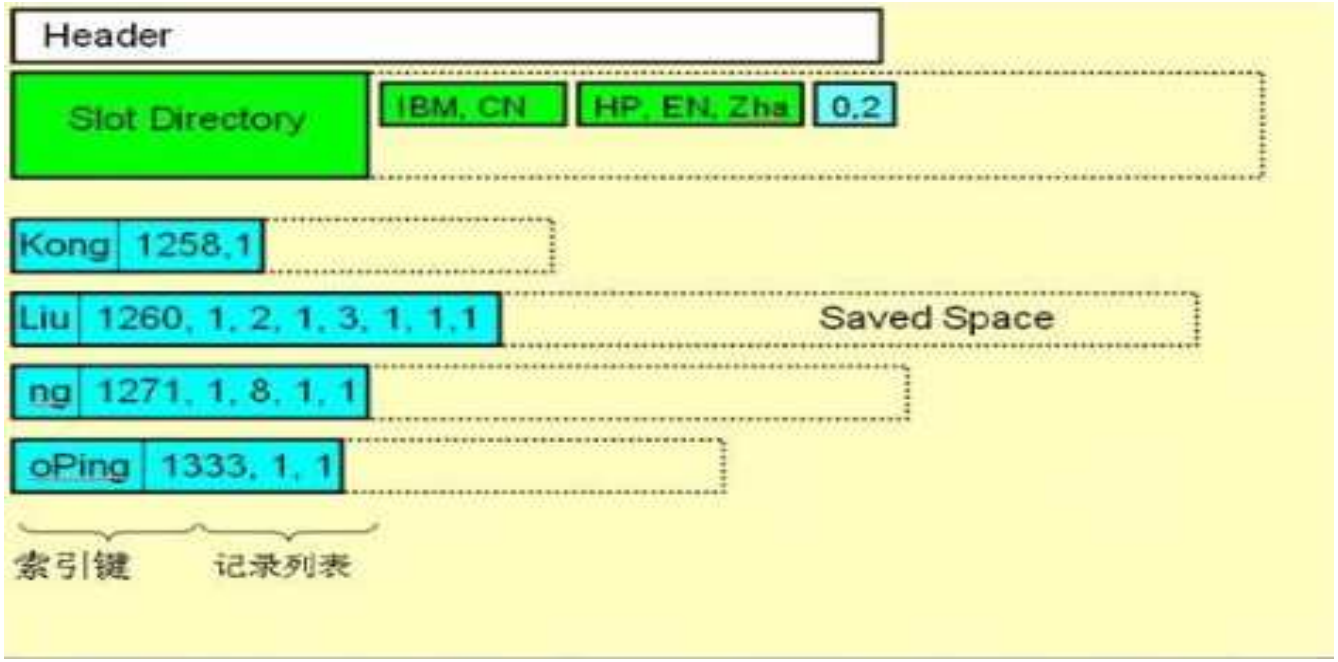
DB2 V9.7索引压缩原理举例



DB2 V9.7索引压缩原理举例



❖ DB2 V9.7索引压缩原理举例



本节讲解内容——杨博斐

1. 变长索引压缩

2. 基于倒排表大小的变长压缩

变长索引压缩

- **Moffat**和**Zobel**也使用了倒排表中的差值。他们充分利用了一个事实：对于大多数长的倒排表，两个记录项的差值相对比较小。

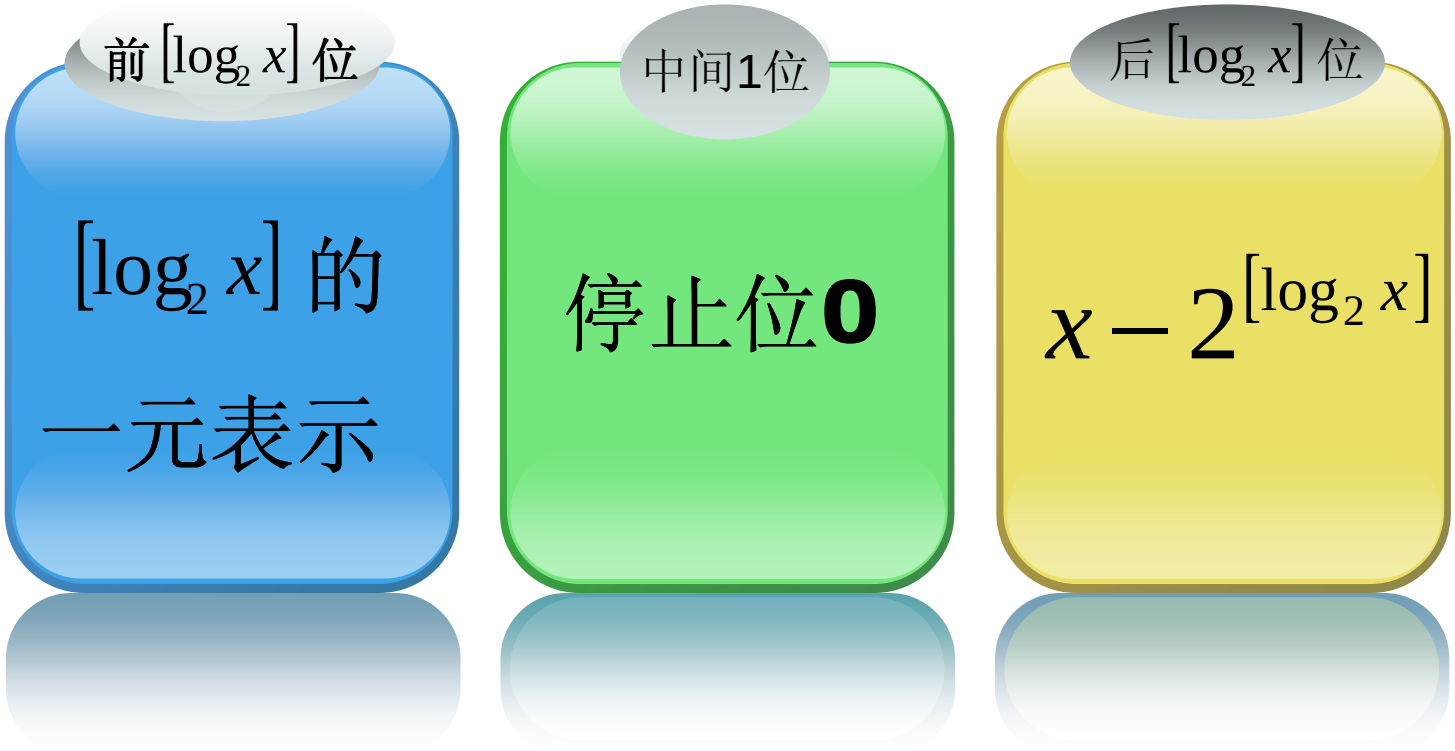
变长索引压缩

基本思路

- 第一次扫描
- 得到偏移量的频率分布规律
- 确立一种压缩模式
- 用新的扫描算法第二次扫描

变长索引压缩

编码：使用 $2[\log_2 x]+1$ 比特来表示整数**x**



注：一元表示是一种基于**1**的整数表示方式，这种表示方式仅使用数字**1.**

如数字 5_{10} 表示为 11111_1

变长索引压缩

- 举例：**10**进制数**14**的压缩
  - $[\log_2 x]=3$ 表示需要一元表示法的111来表示
  - 接下来一位是停止位
  - 剩余的部分 $x-2^{[\log_2 x]}=14-8=6$，由 $[\log_2 14]=3$ 得到，所以使用3个比特位，即二进制110.

因此，14_{10} 压缩后的编码为**1110110**

变长索引压缩

- 由于我们知道数字停止位前的**n**位，这样解压只需要扫描一遍，停止位后也有**n**位。前**8**位使用表中给出的**Elias γ** 编码。

x	γ
1	0
2	100
3	101
4	11000
5	11001
6	11010
7	11011
8	1110000

变长索引压缩

❖ 变长压缩举例：

值	压缩的比特串
1	0
2	100
4	11000
63	11111011111
180	111111100110100

- 这样编码仅需要35个比特。与简单地BA压缩相比，它少用了13比特。
- 使用 γ 编码来压缩倒排表中的词频

基于倒排表大小的变长压缩

- 基于向量V的编码范式：
  - V由i个正整数组成，且 $\sum v_i \geq N$
  - 找出满足下式的k值，对k值进行编码

$$\sum_{j=1}^{k-1} v_j < x \leq \sum_{j=1}^{k} v_j$$

  - 后面跟着差值：

$$d = x - \sum_{j=1}^{k-1} v_j - 1$$

基于倒排表大小的变长压缩

❖ 举例**1**：以数值**7**为例

➢ 使用向量V<1,2,4,8,16>.

➢ 前3个分量（1,2,4）的和等于或者超过7，所以k=3。

k-1用一元表示法为11。

➢差值 $d = 7-(1+2)-1 = 3$。进行 $\lfloor \log_2 v_k \rfloor = \lfloor \log_2 4 \rfloor = 2$ 比特的编码。即11。

因此，7_{10} 压缩后的编码为**11011**。

基于倒排表大小的变长压缩

- 显然，**V**可以根据所给的不同压缩特性进行改变。
- 对于每一个倒排表使用不同的**V**。

  V=<b,2b,4b,8b,16b,32b,64b,…,>,其中**b**是倒排表中已知偏移量的中位数。

基于倒排表大小的变长压缩

❖ 基于倒排表大小的压缩举例：

差值分别为1,2,4,63,180，中位数为4。

所以向量V=<4,8,16,32,64,128,256>

值	压缩的比特串
1	000
2	001
4	011
63	11110000010
180	1111100110111

- 这种方案需要33个比特。在本例中使用的中位数并不是一个好选择，因为数字的改变比较大
- 在比较典型的倒排表中，由于数字均匀地靠近中位数，所以可以得到更好的压缩效果

基于倒排表大小的变长压缩

- 基于吞吐量优化的压缩：

Ahh和Moffat研究出一种保证在有效地查询处理时快速解压，而且压缩率相当高的索引压缩方法。他们研究出一种变长编码方法，可以充分利用每一个倒排表中文档标识符偏移量的分布规律。这种方法综合了位对齐压缩和字节对齐压缩。

基于倒排表大小的变长压缩

❖ 基于吞吐量优化的压缩：

- 选择器部分：包含一个字间划分策略表的索引
- 数据存储部分：每个整型数据编码使用相同的比特数

基于倒排表大小的变长压缩

- 三种基于这种策略的不同方案：
  - **Simple-9**：使用28比特来存储数据，4比特来存储选择器域。选择表有9行，也就有9种不同方式来相等地划分28比特。
  - **Relative-10:** 与Simple-9类似，但是仅使用2比特来存储选择器域，剩下30比特有10种划分方法。主要不同在于：仅使用2选择器比特，每个词仅能从10种可用划分中选择4种——与先前词的选择器的值有关联。与Simple-9相比，这种算法有一定的提高。
  - **Carryover-12:** 这种方法是对Relative-10的一种改进。由于划分浪费的一些空间通过将未用完的比特用来存储下一词的选择器值进行重新利用，这样就可以使用所有的比特来存储下一个字。这种方法是3种方法中压缩效率最高的，当然也是最复杂的，需要更多的解压时间。

基于倒排表大小的变长压缩

- **Simple-9举例：**

划分28比特数据的9种不同方式（使用Simple-9）

选择器	编码	长度（比特）	未使用比特数
a	28	1	0
b	14	2	0
c	9	3	1
d	7	4	0
e	5	5	3
f	4	7	0
g	3	9	1
h	2	14	0
i	1	28	0

基于倒排表大小的变长压缩

- **Simple-9举例：**

继续使用前面的例子，差值分别为1、2、4、63和180。根据选择表，找出合适的编码方式。本例应使用选择器f。这样，对于前四个偏移量，就得到了4个7比特的编码（如下表），其中行f的编码为4位（例如0101）。

Simple-9示例

值	压缩的比特串
1	0000000
2	0000001
4	0000011
63	0111110

基于倒排表大小的变长压缩

- **Simple-9举例：**
  - 这个例子中的最后一个偏移量180，在前32为的词的表示范围内不能表示。因此，我们需要增加一个32比特来进行编码。这样，使用Simple-9来对我们的例子进行压缩，总共需要64位。
  - 这个例子表明：这些压缩方法对于较长的倒排表是最有效的。
  - 此外，由于每个词的每个数值编码为固定长度，这些方法的解压时间非常快。

块寻址的索引压缩

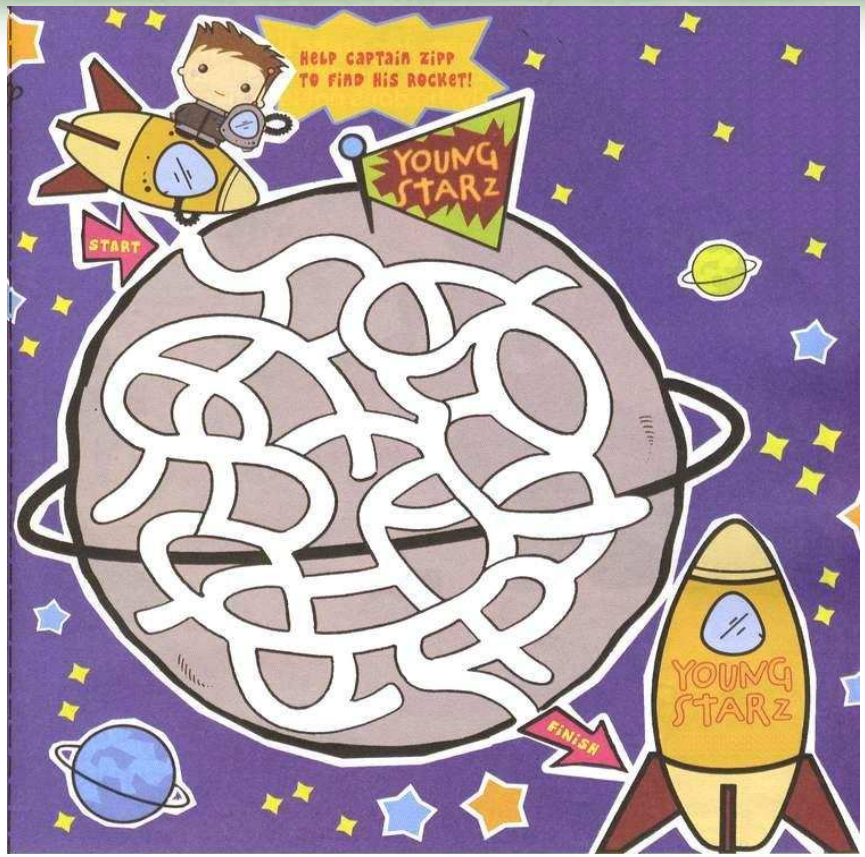
另外一种减少索引大小的方法是创建一个可以对固定大小的文本块进行寻址的索引。

这可以均衡考虑创建索引的时间和用于提高索引与查询处理速度的存储方式。

本节讲解内容----吴玥

1. 索引剪枝
2. 重新排序文档
3. 倒排索引的修订

索引剪枝



- 走迷宫的思路

1、这个方向有路可走

2、往这个方向前进

3、是死胡同，往回走

4、重复第一步，直到找着出口。

剪枝，就是避开死胡同，将不能到达我们需要的解的枝条“剪”掉，以减少搜索的时间。

索引剪枝

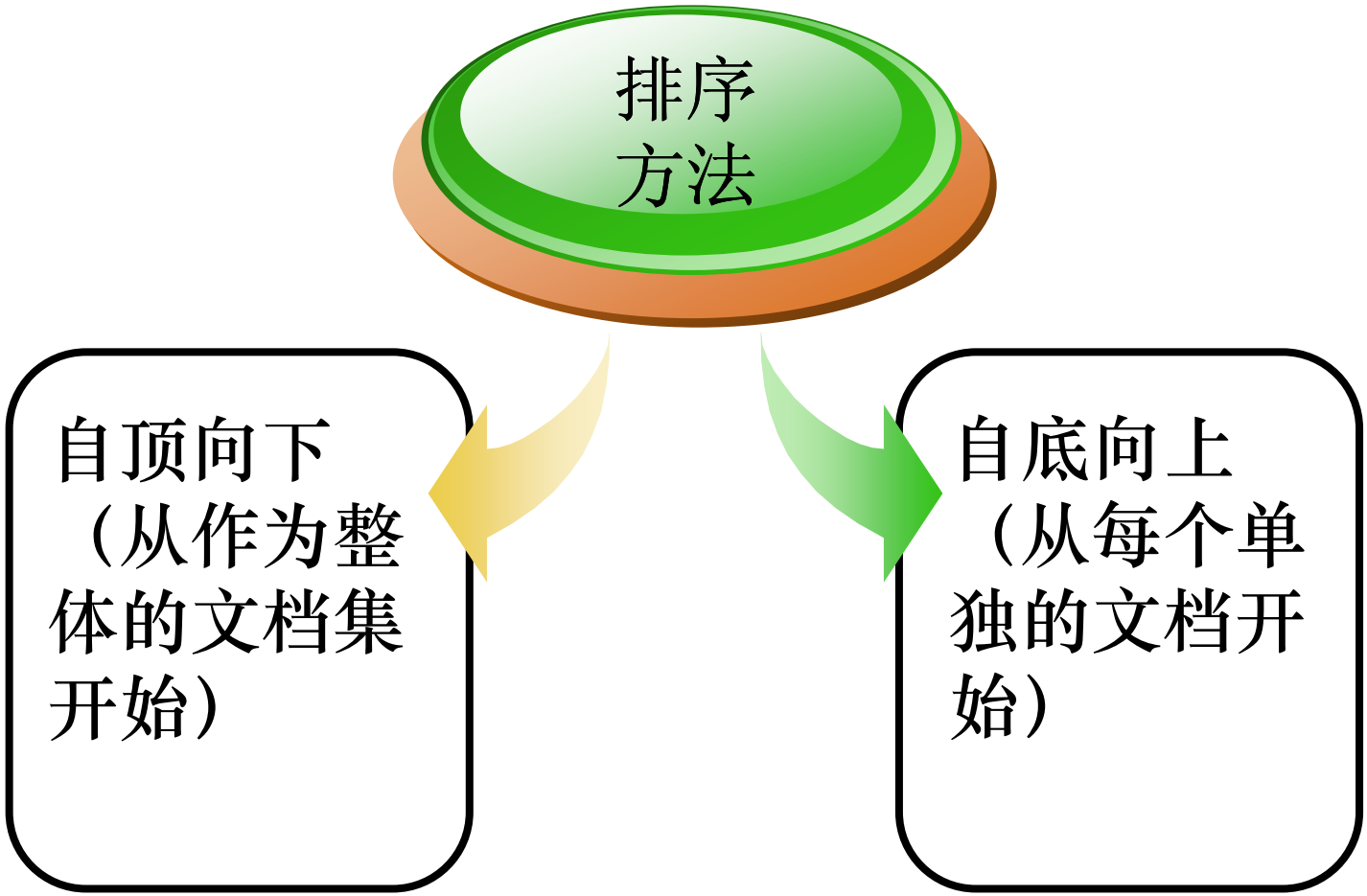
- 静态剪枝使用统一的方式简单地去除倒排表中的记录项——不考虑索引项。
- 文章中的 “in”, “once” ,“too”等词没有什么实际意义
- 中文中的“的”“是”等字通常也无具体含义，这些不代表概念的词可以过滤掉
- 实验表明，在**70%**的倒排索引上进行分层次的剪枝不会显著影响平均准确率

在构建索引前对文档重新排序

提升索引压缩的效率——在压缩倒排索引前对文档重新排序

- 以文档*d*1，*d*99和*d*1000为例
- 包含相同的索引词*t*。
- 索引词*t*对应的倒排表记录项——*d*1, *d*51, *d*101。
- 按照*d*1,*d*2,*d*3的顺序提交这些文档，这样就完全消除文档的间隔。
- **基本思想——相似的文档将包含相似的索引词**

在构建索引前对文档重新排序



自顶向下

幸运52节目中的猜商品价格的游戏，要求你在最短的时间内猜出商品的价格

- 一、选择中心点
- 二、重新分配
- 三、递归阶段
- 四、合并阶段

自顶向下

- 第一种自顶向下的算法被称作事务型B&B
- 第二种自顶向下的算法被称作二分法
- 二分法的中心选择阶段由随机选择的两篇文档作为中心

自底向上

- 文档集中的每篇文档是独立的，并且它们基于相似度逐渐聚成组。
- K-Means算法：依据相似度将其他的文档赋值给标准类，并不断变换标准类
- k-scan算法是single-pass的K-Means算法的一种简化版本 ——相似度最高原则

倒排索引的修订

- **Moffat和Zobel已经证明：通过修改倒排索引来支持快速倒排表扫描，可以提升查询性能**
- 首先处理词频最小的词
- 为了避免扫描非常长的倒排表，修正后的算法如下：
- 对于查询Q中的每一个词t
- 　　获取包含词t的文档的倒排表p
- 　　对于在倒排表中d篇文档中的每一篇文档x
- 　　扫描倒排表p找到文档x
- 　　如果x存在
- 　　更新文档x的得分

以google为例

- 我们假定倒排表中的每条记录是按文档标识符排序的
- 每个桶（barrels）保存了一定范围内的wordID
- 两个倒排桶集合——一个集合只包括标题和锚命中（后面简称短桶），另一个集合包含所有的命中（后面简称全桶）。

以google为例

- 1解析查询（Query）。
- 2把单词转化成wordID。
- 3从每个单词的短桶文档列表开始查找。
- 4扫描文档列表直到有一个文档匹配了所有的搜索词语。
- 5计算这个文档对应于查询的评分。
- 6如果我们到达短桶的文档列表结尾，从每个单词的全桶(full barrel)文档列表开始查找，跳到第4步。
- 7如果我们没有到达任何文档列表的结尾，跳到第4步。
- 8根据评分对匹配的文档排序，然后返回评分最高的k个。

以google为例

- 评分系统：综合考虑了锚文本命中和页面的PageRank值。
- Google将命中分为不同类型
- 每一种类型都有自己的类型权重值（type-weight）
- Google数出命中列表中每种类型命中的数量，每个数量转化成一个数量权重（count-weight）
- 通过数量权重向量和类型权重向量的点乘为一个文档算出一个IR分数。最后这个IR分数与PageRank综合产生这个文档最终的评分。

本节讲解内容——谷雨

1. 部分结果集检索

2. 简化向量空间

度量倒排表特性的种类

1. 基于文档频率的裁剪

25%~75% ⟶ 避免噪音词

2. 基于最大估计权重的裁剪

$tf_{\max} \times idf$ 假定查询词的最大词频

$f(I) = i^e, 0 < e < 1$

3. 基于倒排表中磁盘页权重的裁剪

$tf_{\max} \times idf \times f(I)$ 假定给定页中的最大词频

按倒排表使用的磁盘页排序，而不是整个倒排表。在创建索引时，倒排表按索引词频的降序存储，并且索引包含倒排表中的每一页的记录，不仅仅包含指向第一级路的指针。
记录项记录一个给定页的最大词频。

简化向量空间

余弦系数公式：

$$SC(Q, D_i) = \frac{\sum_{j=1}^{t} w_{qj} d_{ij}}{\sqrt{\sum_{j=1}^{t} (d_{ij})^2 \sum_{j=1}^{t} (w_{ij})^2}}$$

w_{qj} 查询向量q中的相应词j的权重

d_{ij} 表示词 t_i 与 t_j 之间的距离

w_{ij} 查询 j中词语 i的权重

简化种类

$$SC(\mathrm{Q},\mathrm{D_i})=\sum_{j=1}^{t} w_{qj} d_{ij}$$

$$SC(\mathrm{Q},\mathrm{D_i})=\sum_{j=1}^{t} tf_{qj} tf_{ij}$$

tf_{ij}表示词j在文档i中出现的次数(权重)

$$SC(\mathrm{Q},\mathrm{D_i})=\sum_{j=1}^{t} w_{qj} w_{ij}$$

如果词j在查询中出现，则权重w_{qj}为1，否则为0

如果词j在文档中出现，则权重$w_{ij}=idf_j$，否则为0

$$SC(\mathrm{Q},\mathrm{D_i})=\sum_{j=1}^{t} w_{qj} w_{ij}$$

如果词j在查询中出现，则权重w_{ij}为1，否则为0

如果词j在文档中出现，则权重w_{ij}为1，否则为0

简化种类

四种简化方法的比较：

基于TREC（文本检索）的子集实验

- TREC描述信息（长查询）

  使用归一化公式计算较好

- TREC概念（短查询）

  使用后两种方法较好，目标性强，准确性高。


Thank You !
小组成员：徐才舒（2120101775）
杨博斐（2120101778）
吴　玥（2120101774）
谷　雨（2120101735）
Contact
Email: kevinzhang@bit.edu.cn
Welcome to visit my blog
http://hi.baidu.com/drkevinzhang/
LOGO
《网络信息内容安全》讲义/张华平/2010