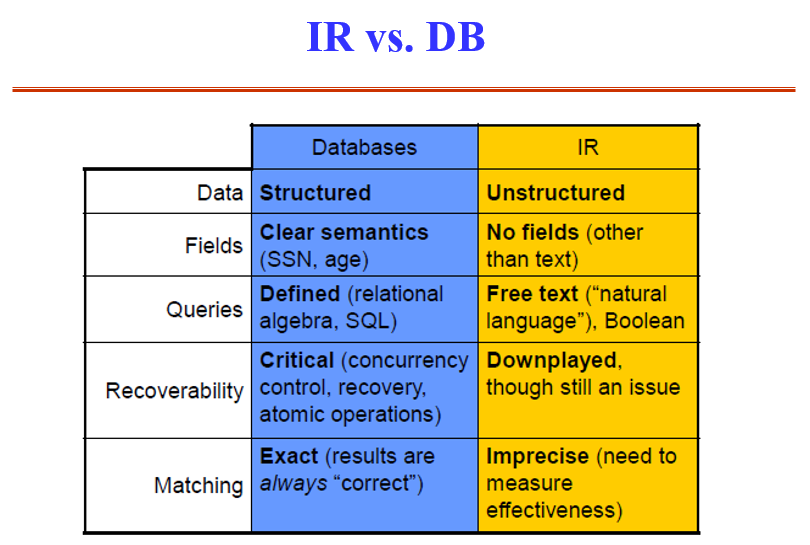
信息检索是对非结构化（相对于数据库而言是非结构数据，算是弱关系）的数据的特征匹配，这也是不同于数据库的地方。

一般就是对文字的检索，将文章中的词的特征提取出来（关键词），处理以后构建一个index搜索进行存储，在用户进行查询的时候提取用户Query语句的关键词并进行处理，在自己构建的index database中进行搜索，寻找匹配的document对象，最后将结果返回给用户。

难点在于对输入的处理：Tokenization、Correction

通过字来“知道”意思

1. 用户输入的不规则性Free Text queries，很可能是按照自己的语言习惯进行检索。所以中文的话就涉及到自然语言处理中的分词来查找关键词。英文的话虽然默认使用空格将用户的输入进行分隔，但是万一有的人喜欢使用逗号进行分隔这就很糟糕。
2. 更可能发生的事情是用户输入的错误，可能是拼写上的错误，对于中文则只能借助可能的上下文进行判断是否存在错误。

**Tokenization：**对于英文输入（匹配的时候全部进行小写化处理）怎么划定一个词

Dave’s -> Dave 去除’s（stem）

O’Neill->Neill 这是人名问题，暂不知道怎么解决

co-education->coeducation 单词中间的-可以直接去除，这个基本都是一行末尾进行单词的连接

U.S.A->USA Vs.->Vs

数字中间的-（电话分隔符）、 ,（数字表示，每三位数字一个逗号）、 /（日期）、货币符号。不能丢了小数点

对于其他情况下的标点符号默认相当于空格

处理“stop world”:of、the 这种大部分情况下没有作用的单词。现代的检索系统已经也将stop world考虑进去了，因为有时候还是挺有用的。比如莎士比亚的一句名言：To be or not to be.

**Correction:** 词的意思与词的样子

不同语种：colour与color 德文中的单词上标

同义词car=automobile

单词变形：is、are、was、were、been、being与be

如果使用词根来做到纠正，比如说将replacement变成replac，那么用户查询的精准度会进一步下降。因为查replace与replacement这两种不同的词性，很显然有着不同的用处的词，却会在不管什么查询的时候，有着一样的输出。

真正纠正两种方法是1.通过上下文环境（上下相关词汇出现的概率）判断这里可能的词汇2.结合与这个错误词汇相近的Edit Distance，通过dp算法直接进行寻找接近的字符串或者通过拆分（对于like进行3-gram拆分就能得到lik与ike）判断相同程度。

优化：

要想加快处理，就先处理query 中df小的term。在posting list中将tf按照从大到小进行排序。只输出K-top resullt，从而快速略去不必要的计算。

丢失结果的质量优化：

只考虑高idf的query term

只考虑包含多个query term的doc

对于doc质量的判断，除了有Query中term的相关性，还有域名中的信息，以及link之间的相互连接关系等几百个因素的加成（向服务器提交一些隐藏信息，比如ip就可以判断出大致的城市，检索的时候就考虑城市）。所以也有推荐按照非相关性因素的影响大小进行排序。

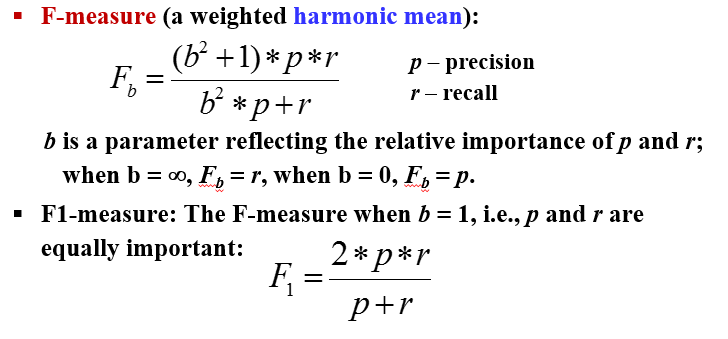
对返回结果评估：准确率与查全度

recall = 查询返回的相关结果/全部数据中的相关结果 随着查询数量的增加不会减少

precision =查询返回的相关结果/查询返回的总结果 随着查询数量的增加波动下降

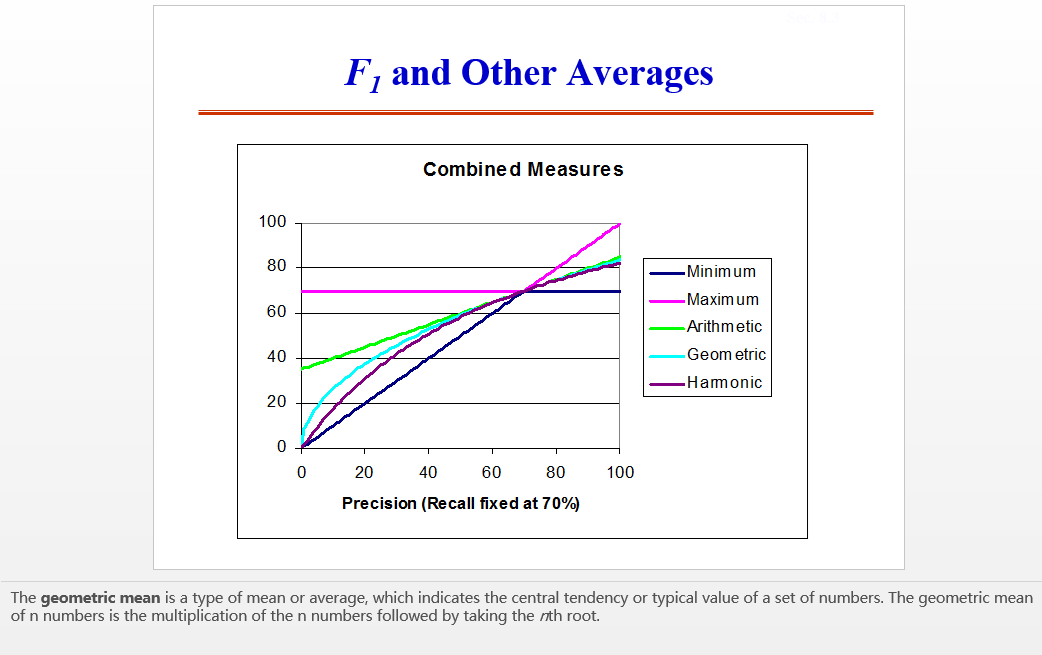
accuracy =（查询返回的相关结果与没有返回的不相关结果）/总

公式：



 调和平均数两种形式

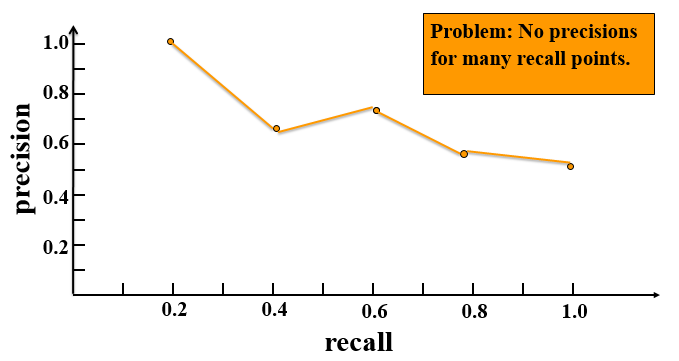
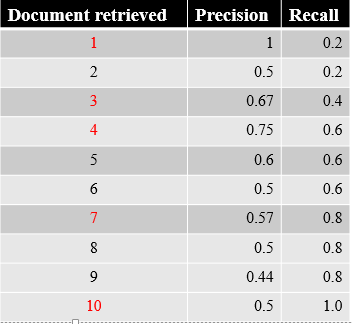
取b=1的时候也就是调和平均数，很适合这种情况，只有当两哥变量都比较大的时候才会整体数值大。而算术平均数很明显在一大一小的时候会有偏移。



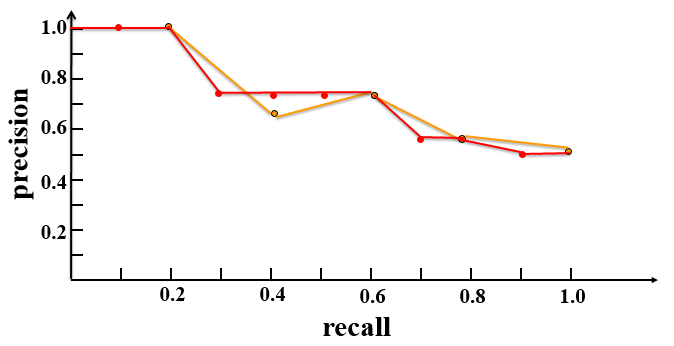
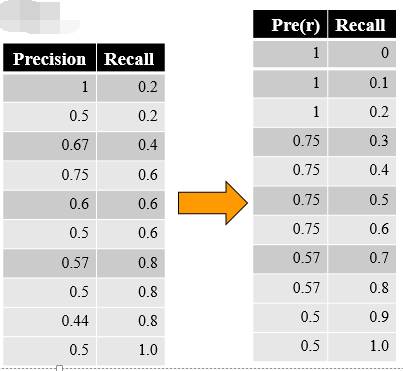
对经过排名以后的结果进行评估：

***11-point recall-precision curve* 从recall、precision引申过来，使用recall横坐标是0, 0.1, 0.2, …, 1.0的十一个点进行绘图，precise是纵坐标。**

下图中红色表示返回结果中的相关查找



**After Interpolated precision cannot go up when recall increases绘制不降图.于是也有了0这个点，凑成了十一个点**



**11-point interpolated average precision ：不使用图表而使用数字**

**(p0 + p1 + … + p10) / 11**

**下面全是对precise的升级；**

**MRR – mean reciprocal rank of the first relevant document.**

****

where *ranki* is the rank position of the first relevant document for the *i*-th query in a set of queries Q. **只是对第一个有关文档出现位置的利用，结果很有局限性。**

**Precision-at-*k* (P@*k*) ：MRR的升级版**

**P@*k* = *M* / *k***

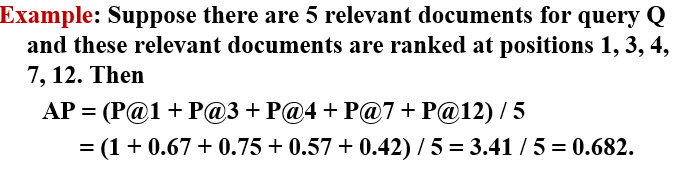
where M is the number of relevant documents among the top k retrieved documents.

**average\_P@k =  除以了Query的长度**

**上述缺点是没有考虑位置，仅仅对数量上的考虑**

**Average precision (AP)： P@K的升级版 取了所有相关点的precious从而尝试将位置考虑进去**

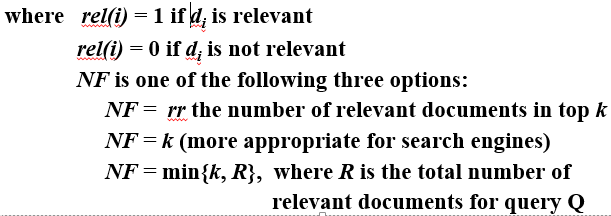




2÷3=0.67 3÷4=0.75…

**Average Precision at cutoff *k* (AP@*k*)： AP取了所有的点而将位置考虑进去**

****



**Mean average precision (MAP) average of the AP of each query in the set of *N***

****

**Mean Average Precision at *k* (MAP@*k*)**

…

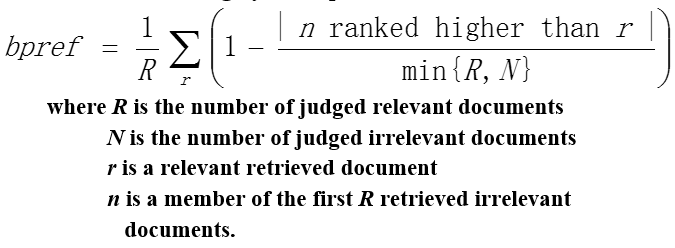
**Geometric mean average precision (GMAP)**

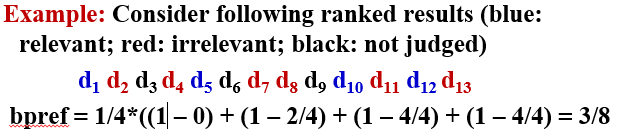


很明显GMAP比MAP好，但是对计算机的运算量要求更加大了。more sensitive to the performance of each query

**bpref: Computes a preference relation of whether *judged relevant documents* are retrieved ahead of *judged irrelevant documents*.**

Designed for situations where relevance judgments are known to be highly incomplete.



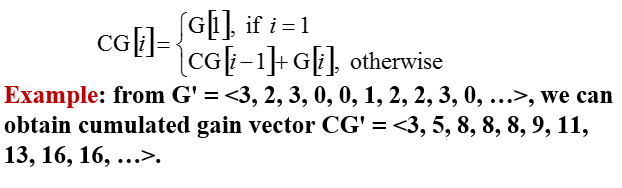


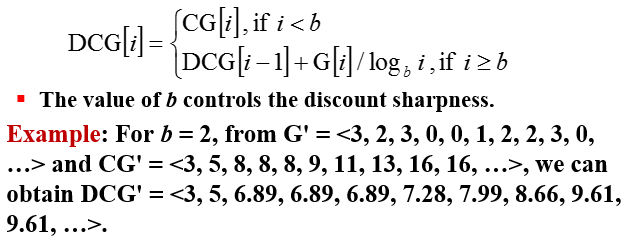
注意最后一个还是4/4，这时候不相关的文档数目已经比相关的文档数目多了。

**NDCG – *Normalized Discounted Cumulated Gain* 归一化折扣累加增益**

**允许通过数字指定设置搜索结果相关程度的等级** i从一开始！

**Gain vector G' = <3, 2, 3, 0, 0, 1, 2, 2, 3, 0, …>** 数字表示相关程度

**Cumulated gain vector** 

**Discounted cumulated gain vector** 

**Ideal CG and Ideal DCG vectors**

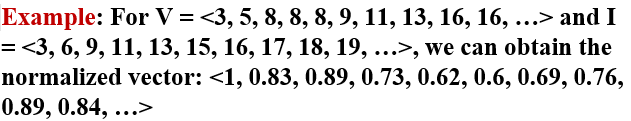
如果说是一个简单的DCG，那么将一开始计算的G’改成I’ ，这是因为显示结果经过了降序排序，那么I**'** 向量数值从左到右就是不增变化。所以这才是理想状态下的结果显示。

**Average vector 对于一个Query集合而言**

For a given query set and a ranking method, let **V** = <v1, v2, …, vk> denote the average (D)CG vector and **I** = <i1, i2, …, ik> denote the average ideal (D)CG vector.

**Normalized discounted cumulated gain (NDCG) vector**

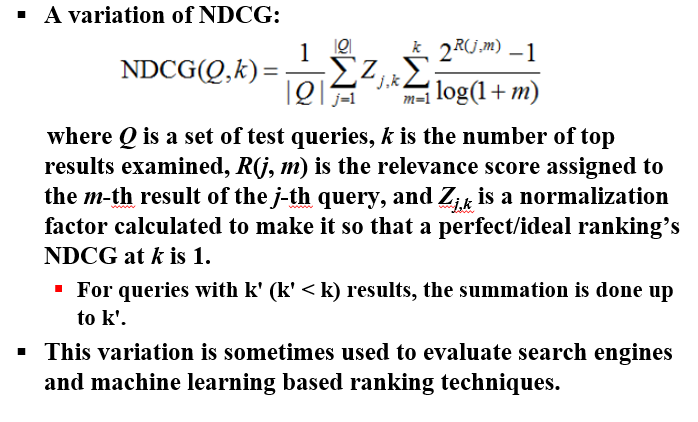
**N(D)CG vector = <v1/i1, v2/i2, …, vk/ik>**



**DCG vector /Ideal DCG vector =nDCG vecotr**

**AN(D)CG@*k* – Average N(D)CG up to the position *k* of the corresponding vector:**

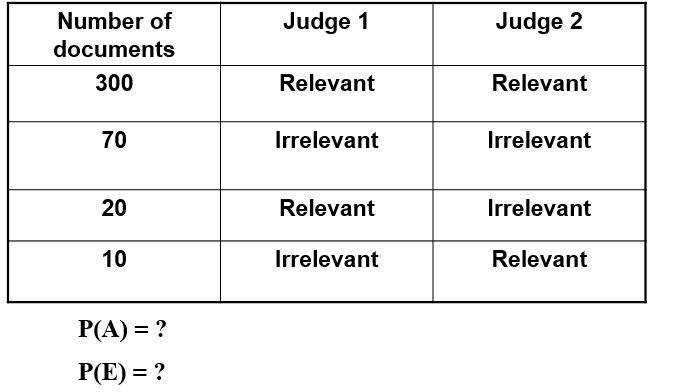
**AN(D)CG@*k*(V) = (Σi=1…*k* V[i]) / *k***

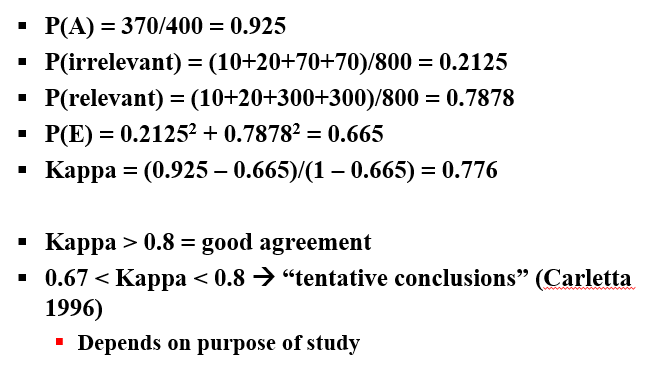


**Kappa = [ P(A) – P(E) ] / [ 1 – P(E) ]**

**P(A) – proportion of time judges agree. 两个人对返回结果中看法相同的数量占比**

**P(E) – what agreement would be by chance 随机选择的时候**





对**Feedback的使用：**

**Rocchio 1971 Algorithm：**

**centroid向量重心公式：**

****

*Dr =* set of known relevant document vectors.

*Dir* = set of known irrelevant document vectors.

通过feedback对向量重心进行偏移，靠近（加上）有关向量，远离（减去）无关向量。、、 这三个变量在一些系统中被限定为大于0。也就是说只允许进行小量的偏移。

**Pseudo Relevance Feedback：**

是否relevance是靠用户进行反馈，但是一般用户并不会进行反馈，所以直接认为在用户query的返回结果中，前部分是relevant，后面一部分是irrelevant。

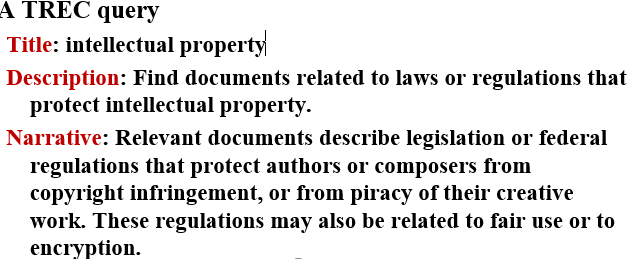
**Implicit Relevance Feedback：**

用户在某几个网页相对停留较久，根据大部分用户的点击量 来表示relevant

下面这个没有明白是干什么用的，似乎是标准：

benchmark设置： 进行基准的设置，包括查询、doc 搜集、结果评估

**TREC (<http://trec.nist.gov/>)**



**GOV2、NTCIR、CLEF**

**A/B Testing： 在进行功能创新的时候将一小部分流量引导到新功能检索界面（与原系统同时运行），借助用户进行公测。通过与原系统进行比较判断这个功能是否值得添加。**

特殊支持：/3表示在3个单词范围之内出现

最简单的模型：Boolean retrieval Mode

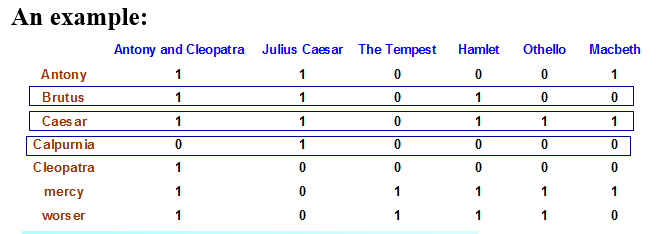
Document-》term-document incident matrix-》inverted index list

User Input-》AND Operation or Or Operation to inverted index list

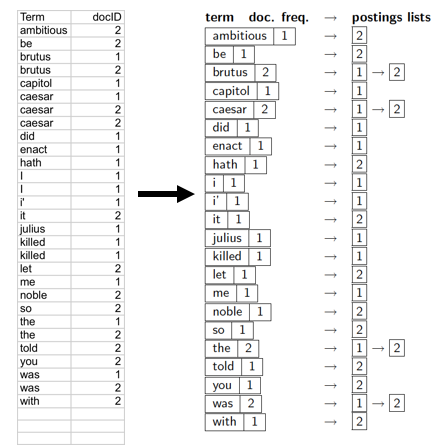
Cons:

1. 搜索结果要么很多要么就基本没有，输出精度不高，同时没有进行ranked排序
2. 对于用户的输入不能进行很好地认知，不能区分“我爱你”、“你爱我”，由于进行单词之间的分界靠空格所以不支持短语搜索
3. 现在的搜索引擎里面记录了term在每一个doc中的位置。这样在返回查找结果的时候可以进行预览。

term-document incident matrix：  
 使用矩阵的表现形式来表示term是否存在在doc。



下图中term表示一个词元， doc.freq表示词元在多少个doc中出现（同样也就是之后的list大小）， posting lists表示具体出现的位置，用于在词元匹配以后将存储的doc作为结果返回。



对list中的doc从小到大排序，我想是为了方便进行去重验证。之所以需要对term进行排序是为了方便在获取用户输入的时候，如果是list（因为数据不够多还要输出到文件）通过二分来匹配找到这个term对应的一系列统计信息。其实也可以使用hash table进行映射。

实现方法：

1. 构建Index Database(hash或者B-、B+ Tree)

遍历文章，将其中的单词（不考虑短语）经过处理作为term进行记录，直接构建term、doc.freq、positings list关系。这时候只要保证遍历文章的时候不是重复的，那么postings list就不需要排序。

1. 连接term与Index Database

这时候可以通过对Index Database中的term进行排序之后使用二分查找，或者直接使用hash table进行映射。

1. 最后处理用户输入，查询Index Database并返回查询到的doc id。

升级：

1. 为了方便对匹配文档进行部分显示，方便用户排查文档是否符合自己的需求，我们在inverted index list中除了记录docID，还要记录其中的每一个位置
2. 一个词在对一篇文档起到的作用与它那篇文档中出现的次数有关系，同时与它在整个文档中出现的频率有关系。这个model中还没有rank系统。

DAAT和TAAT的top-k算法：<https://static.googleusercontent.com/media/research.google.com/zh-CN//pubs/archive/37365.pdf>

向量空间模型：**Vector space Model/bag of words model**

query向量中的每一个元素都是**tf-idf Weighting**

document向量中每一个元素斗士**tf weighting**

计算similarity进行向量点积的时候会将df的weight考虑进去，所以只需要一边有就好了，而query中有是对计算量不错的减小。

之所以会有个bag of words model的别名，是因为“我爱你”和“你爱我”的检索是一样的，就像将word直接装入一个不透明的bag中。计算两个向量之间的相关性使用的是角度cosine的计算公式，同时由于query 向量的大小是固定的，所以在计算表达式的时候可以省略这一步除法。其中向量的乘积运算刚好将无关的term化为0。

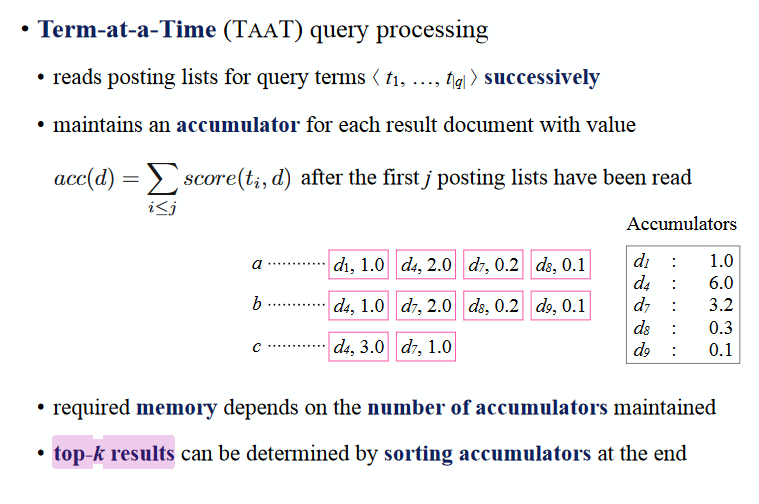
使用：

1. 先通过Boolean Retrival Model 来缩小各个可能Vector的范围，再使用Vector
2. 对posting list 进行排序，这样返回doc进行输出的时候勉强算作有rank。
3. 老师的方法：由于只输K条最匹配记录。

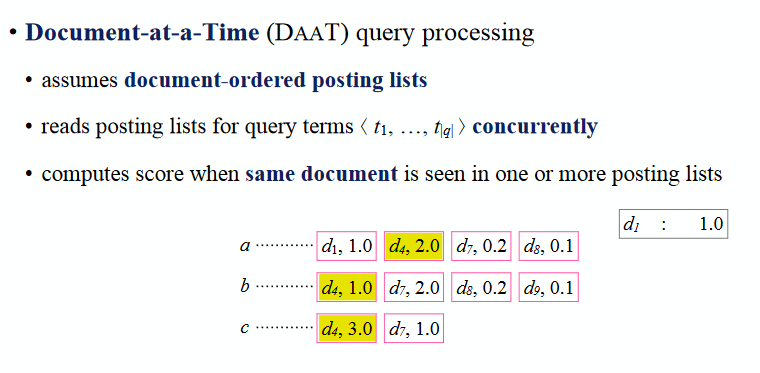
两种实现算法K条与query最匹配记录Top-k retrieval：

直接尝试去寻找最佳匹配的K条记录，找到了就停止。就不一定需要遍历posting-list全部内容。下面是两种主要实现算法：

TAAT Term At A Time ：



DAAT Document At A Time ：



我们提前按照tf从大到小对posting list进行排序，也不会影响这种方法。

概率模型：**Binary independence model**

用概率来表示相关性，排序以后返回结果给用户。必须要对每一个doc都进行概率计算。

**P(*rel* | *Di*, *Q*)** 简写为 **P(*rel* | *Di*) 之后都隐藏这个**

将Query与Document相关性函数取名为**osim**

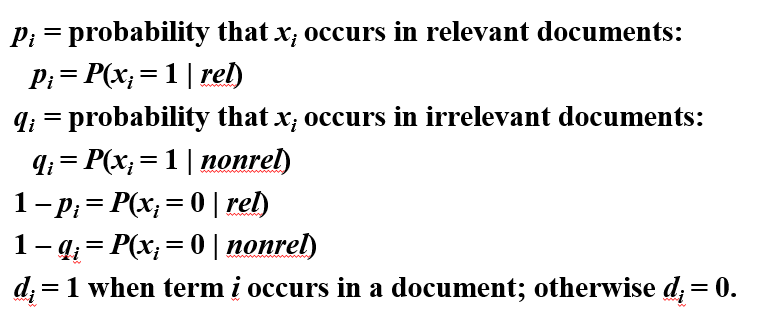
，根据贝叶斯公式 we get 。 So **osim function** is  使用这个函数的值可以进行相关性的比较。

再假定每一个term之间是相互独立的，则有

因为乘法对计算性能有不小的消耗，所以改进使用

则有 。

Let x = (x1, …, xt) denote a set of terms; xi is 1 or 0。1 and 0 indicate the presence and absence of term i in the document.



***ri* is the number of relevant documents containing term *i***

***dfi is the document frequency of term i.***

***R* is the total number of relevant documents for query *Q*.**

***N* is the total number of documents.**

则 ***pi*= *P*(*xi* = 1 | *rel*) = *ri* / *R***

***qi* = *P*(*xi* = 1 | *nonrel*) = (*dfi* – *ri*) / (*N* – *R*)**

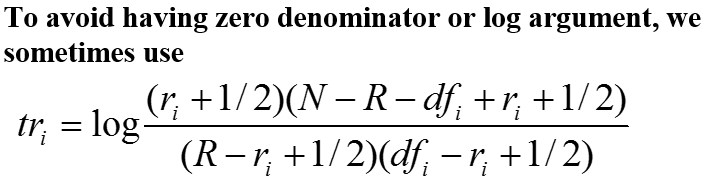
di要么是1，要么是0。



所以doc概率的计算就是看后面那个表达

***term-relevance weight***



进行变量值的估计得到

***retrieval status value* (RSV) of *D:***



RSV(D)函数可以作为一个**osim相关性函数**

**pi 、*qi*值评估*：***

由于相关的doc所占的比例基本比较少，所以进行近似***qi*= (*dfi* – *ri*) / (*N* – *R*) *≈* *dfi* / *N***所以可以**：**

**tri == log(pi /(1 – pi)).+ log((N – dfi) / dfi)**

log((*N* – *dfi*) / *dfi*) is a form of *idf* weight

如果认为***pi = 1/2*** 则**tri == log((N – dfi) / dfi)**

但是***pi* usually increases when *dfi* increases**，所以有人提出：



如果考虑用户的feedback

**the set of relevant documents *VR***

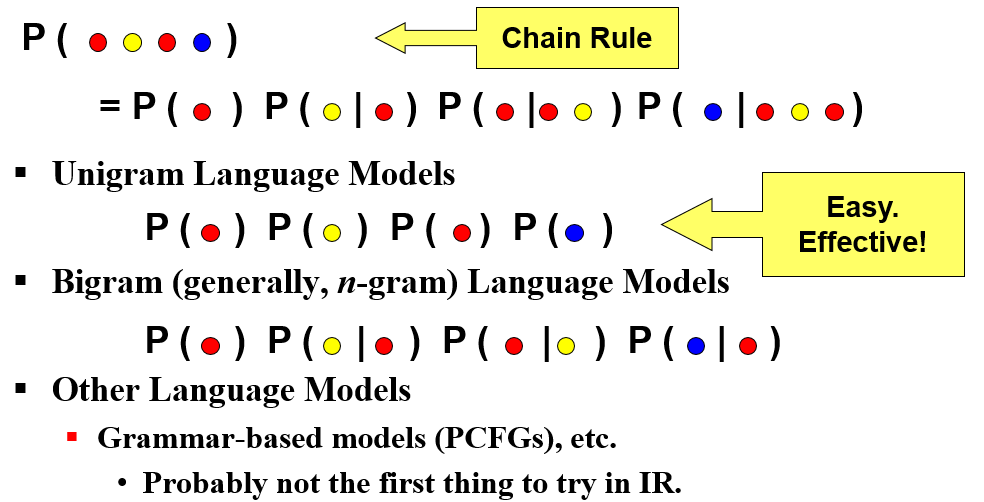
**Let *VRi* be the set of documents in *VR* that contain term *i***

***pi = |VRi | / |VR| or pi = (|VRi | + 1/2) / (|VR| + 1)***

如果认为没有被检索到的是irrelevant，那么

**The Language Model（LM）:**

假装有带有概率的自动机，这个自动机可以产生许多字符串，从而与quert term匹配。为每个文档推荐语言模型，判断由这些产生对应query的概率，根据这个概率排序rank得到输出结果。



Unigram model产生的每一个term之间相互独立，所以doc的概率是:

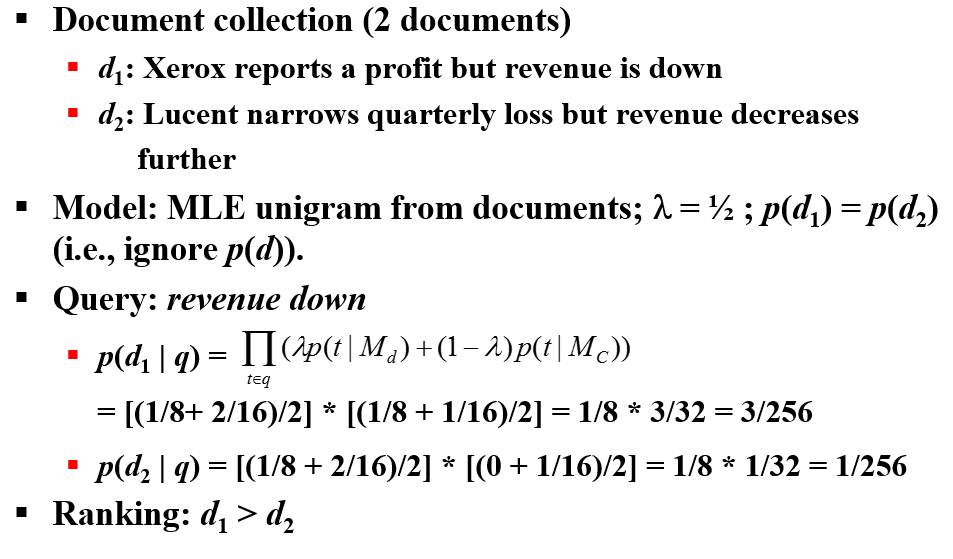
(这也就是模型的建立)

由于贝叶斯规则，可以知道***p*(*doc | q*) = *p*(*q | doc*) \* *p*(*doc*) / *p*(*q*)**，所以就是取决于***p*(*q | doc*)**，即模型产生出这个query的概率大小：

 对与一个给定的query这个系数是常数，所以进行忽略，也就是使用了最大似然估计以后就个概率模型差不多了。

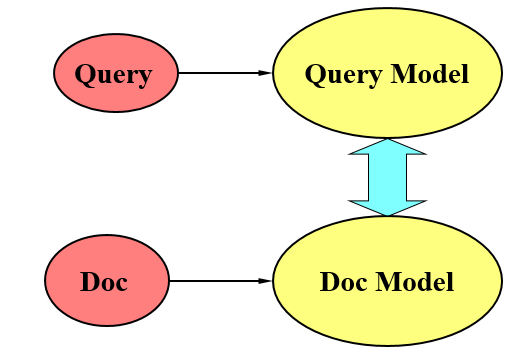
**Mixture Model：不止使用了一种模型（使用了第二种模型进行0概率处理）**

仅仅直接使用来自doc中的term概率是不够好的，所以需要对概率进行smooth处理（最大似然估计以后就需要对0概率进行特殊处理，分出一部分概率进行均分），给每一个元素增加一定的值，然后进行**normalize**单位化亦是一种分出概率的方法。不过这里是使用了p(doc)的概率不进行考虑，让Mc来处理0概率问题，称之为**general language model** 代表着是从所有doc角度来进行考虑。类似于tf-idf权重计算， 当doc包含的df越大的时候p(doc)的值会增加，那么对于所有的含有这个高df term的document而言公式这部分的值是差不多的，这个乘积表达式就取决于其他乘积因子。如果是df小的模块，那么最终的p很受到tf的影响，因为和tf相比它太小了。但是这种类似于idf的效果还是有点不完善，对于查询为“I report”一个doc仅仅有I，另一个仅仅有report，I的df远高于report这个term，但是最终出来的却是rank: I > report。这种idf只对都有一样的term的doc有作用。



λ使得最终计算出来了的概率和是1.

**Retrieval: (1) Query likelihood(就是由doc来产生query的LM), (2) Document likelihood, (3) Model comparison**计算概率的时候，可以由query通过模型机产生doc，也可以doc通过模型机产生query。还有下面这种比较model之间的相似性：

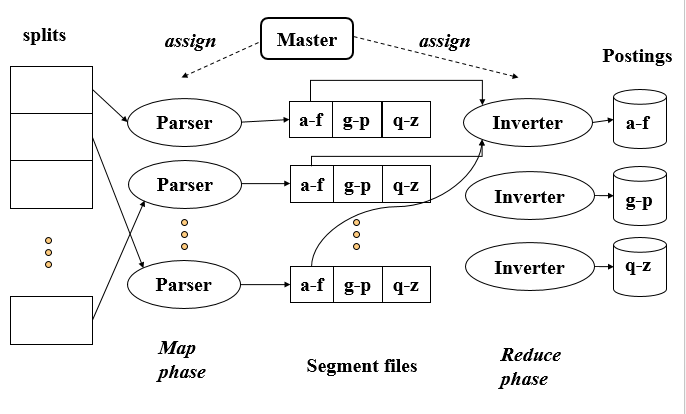


动态变化（增加）的Index Database：

方一、设置一主多辅结构，在合适的时机（尽量不影响用户体验）合并到主库中。每一次查询会将主辅中都查询一遍，进行结构合并以后输出。

方二、通过设置给运行中程序消息，告诉他应该增加内容了，然后读取新增的内容加入到内存中的结构里。

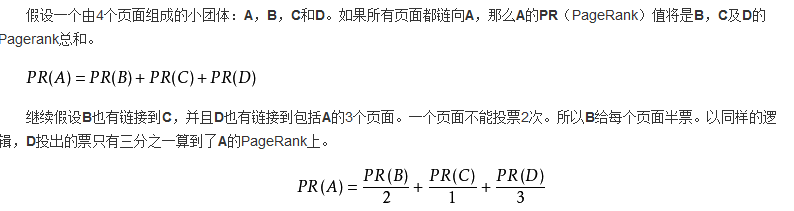
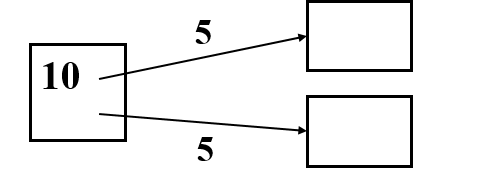
对于分布式系统：



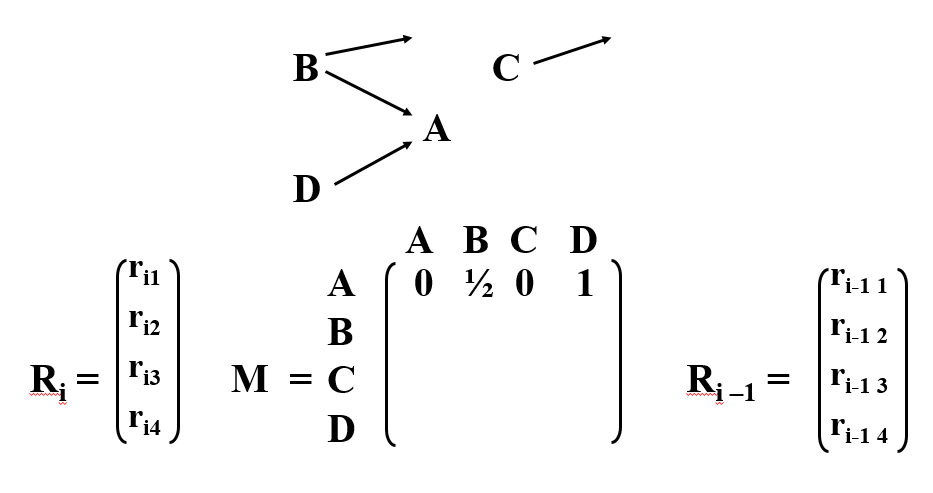
有个词汇叫做MapReduce就是这里面的Map（映射）、Reduce（归约）两个步骤。这是一种程序分布式模型。

通过考虑链接与反向链接**尤其是PageRank投票**来进行网页权重的判断：

比如下面将网页的重要性进行分割然后传递给被链接的网页：

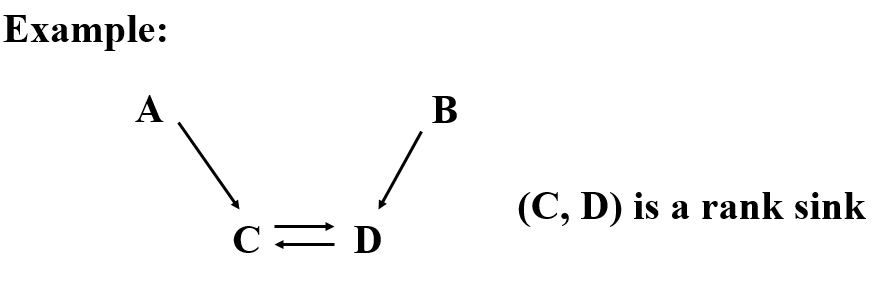


可以通过将一开始所有的网页的权重表示成1/N，然后自动迭代产生数据，逐渐收敛。迭代可以通过矩阵运算：



***Rank sink问题*:**

一个或者一组网页，接受别人的权重传递·，但是不会将自己的权重传递出。



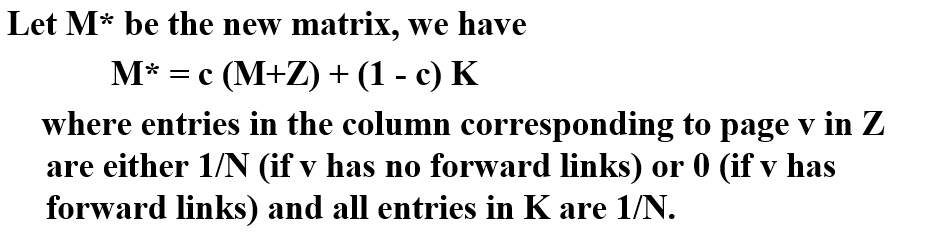
因为每一个网页都只能进行一次投票所以不会出现死循环。通过PageRank可以搜集到高质量网站的名字，比如CSDN，这时候就可以进行权重增加。

***random walk model：这种自动迭代的都是***

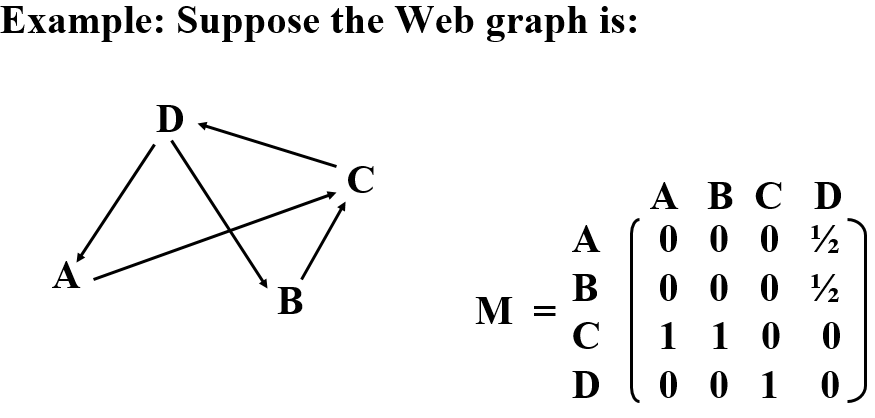
为了防止投票以后，有的页面并没有被投票在矩阵中值为0，所以使用了比例系数d

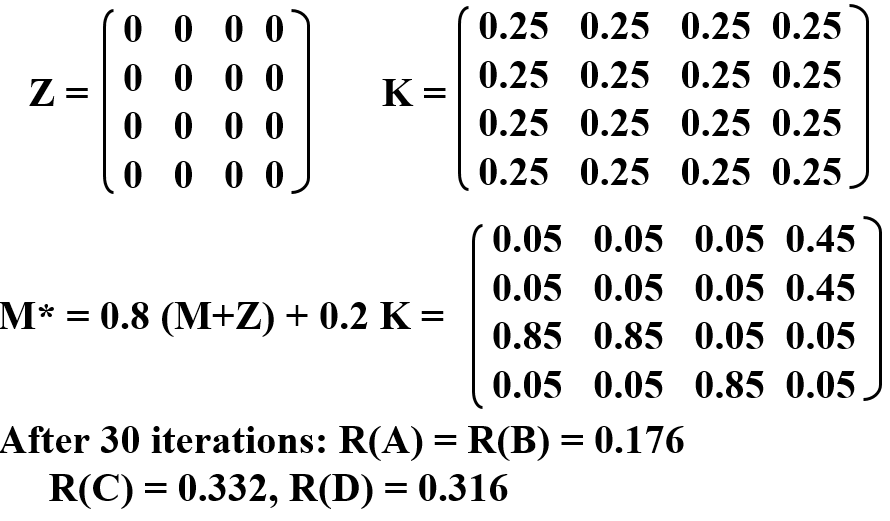
https://gss2.bdstatic.com/-fo3dSag_xI4khGkpoWK1HF6hhy/baike/s%3D380/sign=c9b964c936fae6cd08b4ad693fb20f9e/dbb44aed2e738bd4ed8e37cfa28b87d6267ff9a5.jpg完全没有除以N的必要，而且1-d的分配可以在接受对方投票的时候直接返回（所以如果自己没有链接向别的网页，那么自己的投票就会完全给投票给自己的几个网页，这就是前向链接权重，所以从前就出现了链接多的网页排名高），同时需要在最后对矩阵已有投票数进行计算然后平均分配确保对0的消除。

上面的式子表现成矩阵就是：**M\* = c (M+Z) + (1 - c) K Ri = M\* × Ri-1**

如果没有前向链接，那么就认为这个网页能跳转到任何一个web doc，也就是将投票进行全体均分。如果存在前向链接，那么通过这个前向链接按照系数打折投票，剩余的部分会全体均分，借助矩阵K。其实不必这样子，以为将一个值分散加到1其他所有的网页中是一种耗时间的做法，完全可以直接将这个值无视掉。我觉得网站在PageRank中最终结果是个0并没有关系。

Z矩阵的存在取不同值，就是对有没有前向链接的划分处理。





M\*矩阵是作为右乘矩阵，所以要确保列的总和为1，使得原矩阵类的总和不变。

2005年初，Google为网页链接推出一项新属性[nofollow](https://baike.baidu.com/item/nofollow" \t "_blank)，使得网站管理员和网站作者可以做出一些Google不计票的链接，也就是说这些链接不算作"投票"。[nofollow](https://baike.baidu.com/item/nofollow" \t "_blank)的设置可以抵制评论垃圾。不会参加PageRank

**超链分析算法：**

考虑关键词的时候，不仅仅考虑当前document中的关键词，还考虑链接中的关键词从而 形成rank。比google的pagerank有局限性，不过两者可以配合使用。不过匹配搜索字符串的时候，可以直接借助这些关键词。

Google的PageRank（最广为使用）与李彦宏的超链分析都是已经申请了专利的技术。

判断doc权重的除了内容、关键字、hypertext link还有域名。**<http://www.gamezero.com/team-0/>** 中域名就包含了game。

**The HITS（Hyperlink-Induced Topic Search） Algorithm：**

Authority Page 就是某个领域、话题的高质量网页：知乎、Stack oveflow

Hub page 比如linux中科大源页面这样

HITS算法的目的即是通过一定的技术手段，在海量网页中找到与用户查询主题相关的高质量“Authority”页面和“Hub”页面。算法的基本假设:1. 一个好的“Authority”页面**会被很多好的**“Hub”页面指向；2. 一个好的“Hub”页面**会指向很多好的**“Authority”页面；

Hub与Authority算是doc的两个共有属性，分别表示outlinks与inlinks。

检索原本应该直接返回的匹配结果称为root set，HITS在此基础上扩展出base增加返回结果（root set中的页面应该是包含query term的，它的权重计算就要比之后base中的doc权重影响大），通过迭代一下算法直至收敛,在没有更多可利用信息前，每个页面的这两个权值都是相同的，外部链接可以都设置为1http://img.my.csdn.net/uploads/201209/24/1348485600_7083.jpg。

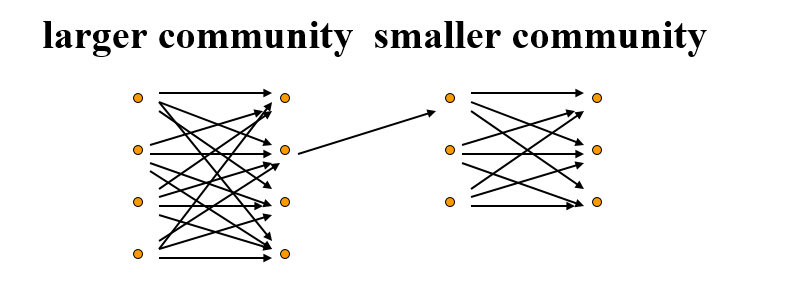
***Transverse link*:** links between pages with different domain names.

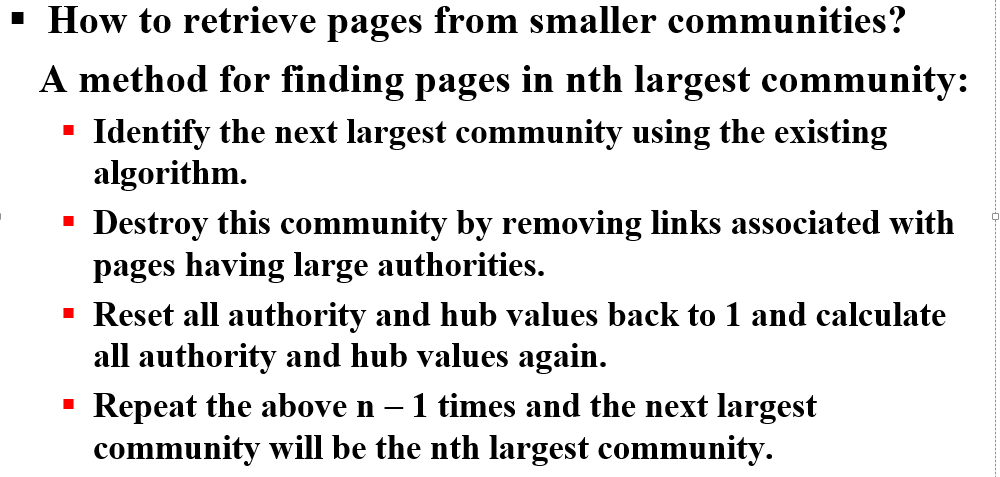
***Intrinsic link*:** links between pages with the same domain name.

Transverse links are more important than intrinsic links.

所以是内部固有链接就给出0-1之间的一个值，而不是1.

大社区的权重也会比小社区高：





网页 a (i)在此轮迭代中的Authority权值即为所有指向网页 a (i)页面的Hub权值之和：

   a (i) = Σ h (i) ;

   网页 a (i)的Hub分值即为所指向的页面的Authority权值之和：

   h (i) = Σ a (i) 。

一个先计算一个后计算，所以不会出现死循环。

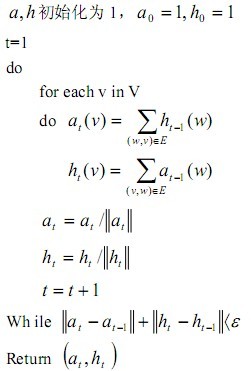
   对a (i)、h (i)进行规范化处理:

   将所有网页的中心度都除以最高中心度以将其标准化：

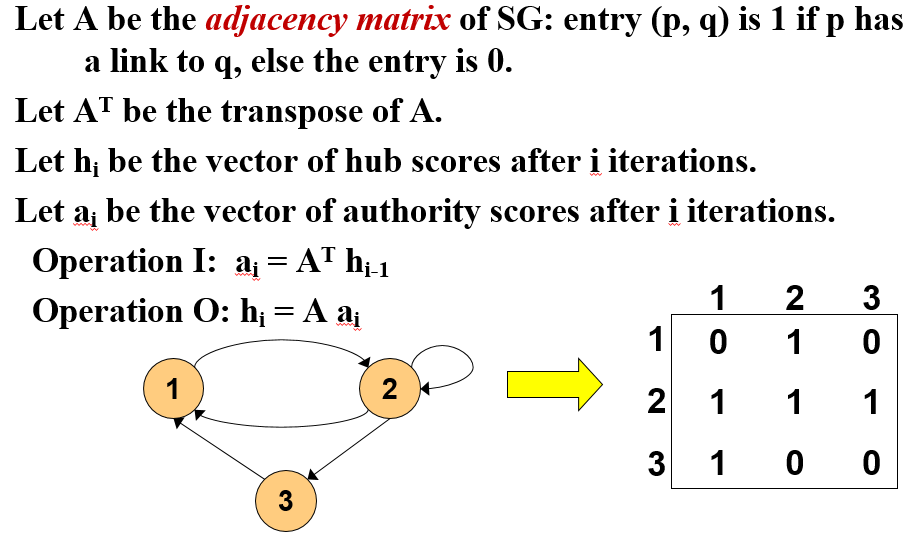
   a (i) = a (i)/|a(i)| ； 似乎就是除以最大值

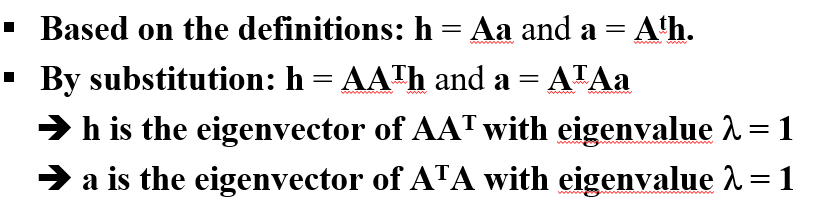
   将所有网页的权威度都除以最高权威度以将其标准化：

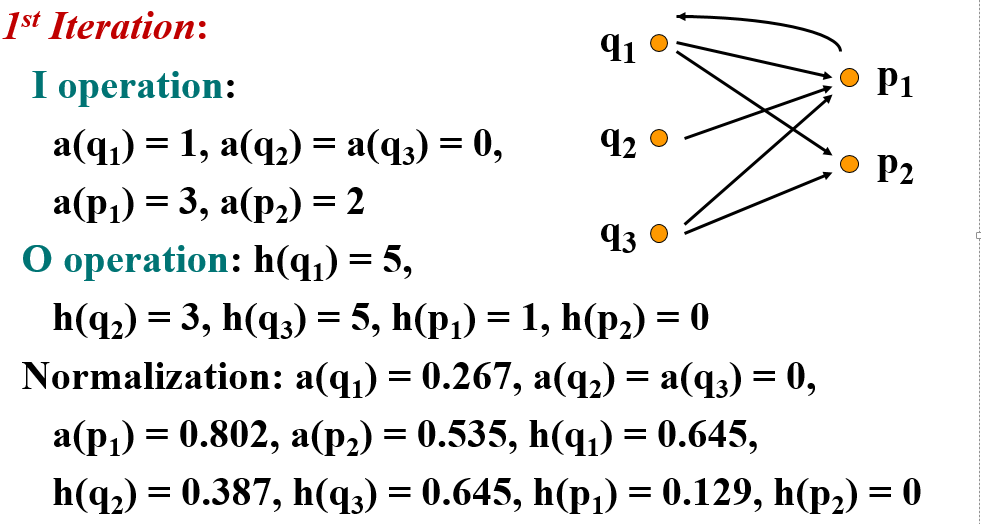
   h (i) = h (i)/ |h(i)|



收敛以后将按照authority score进行排序，然后输出。

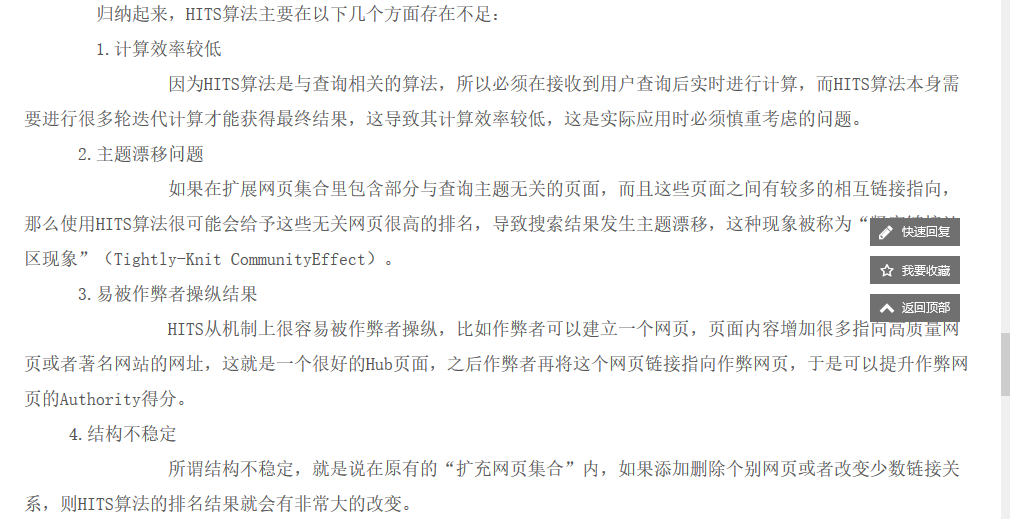






经过多次迭代以后，左边三个的a()是0，右边两个的h()是0.

缺点：

****

从链接反作弊的角度来说，PageRank（不考虑前向链接权重）从机制上优于HITS算法，而HITS算法更易遭受链接作弊的影响。HITS算法结构不稳定，当对“扩充网页集合”内链接关系作出很小改变，则对最终排名有很大影响；而PageRank相对HITS而言表现稳定，其根本原因在于PageRank计算时的“远程跳转”。

**Connectivity Server:**

Support for fast queries on the web graph

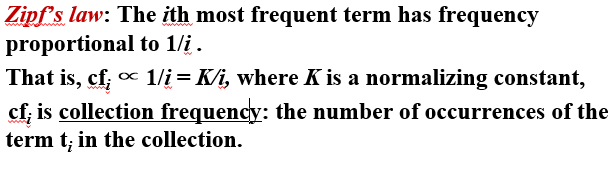
Which URLs point to a given URL?

Which URLs does a given URL point to?

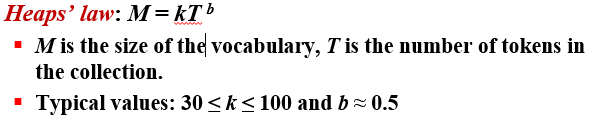
基本目标：在内存中维护一个邻接链表

**定律**

**zipf’s law:** 一个单词出现的频率与它在频率表中的排名成反比关系。等价于说排名与出现的频率的乘积是个常数。



**Headp’s law:** 常常用来计算在一直M的时候，T的数值有多大



**Jaccard Coefficient：**

**|A ∩ B| / |A ∪ B|**

**Dice’s Coefficient**

**2 \* |A ∩ B| / (|A| + |B|)**

上面是描述两集合相关性程度的系数，公式可以认为是一个东西。如果用在Query与Document中的相关性判断，还差了tf。就算加上tf，也还差了每一个term之间的排列顺序。

**Log-frequency Weighting：**

****

**idf Weight：**（df is document frequency）

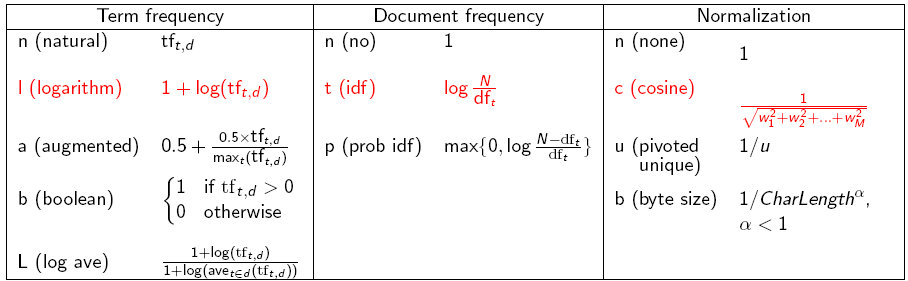
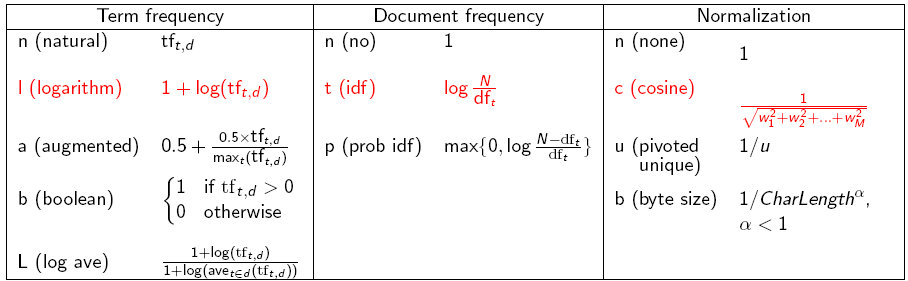
** 取了log是为了乘法好算**

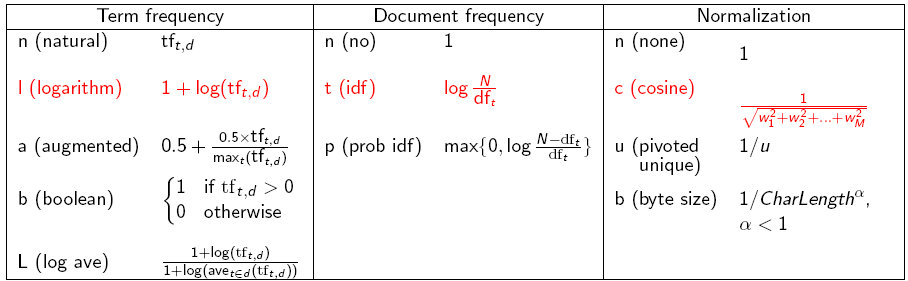
**tf-idf Weighting：**

****



上面是判断term与文章权重的常用公式。还有下面几种，最常用的使用红色标出。



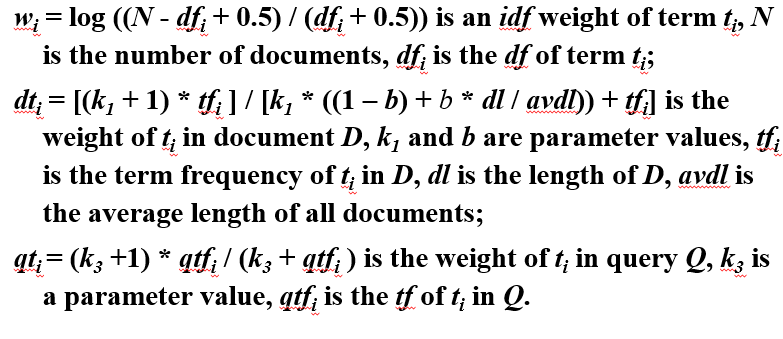


Normalization 就是向量空间中向量的长度单位化。

**Okapi BM25 Function： 这也是种rank方法**

**This function is based on the probabilistic retrieval model。等同tf-idf。**





Wi could be negative if term ti appears in more than half of the documents.

1 ≤ qti ≤ min {k3 + 1, qtfi}

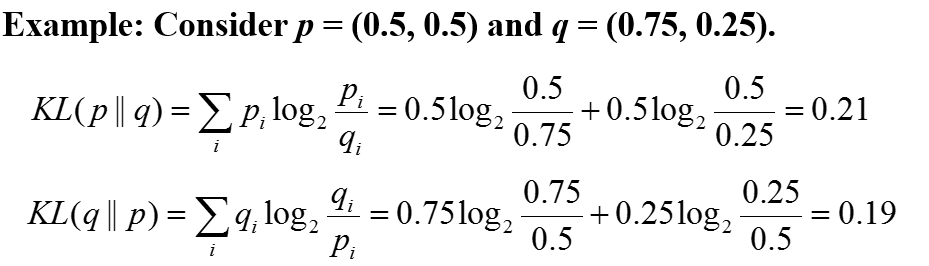
前者越大，说明这个词对这个文档约有作用，后者越高，说明这个词每出现一次起到的作用都很有限。

**Kullback-Leibler (KL) Divergence： 描述两种概率分布之间的近似度，相对熵**

KL散度是用来 度量使用基于Q的编码来编码来自P的样本平均所需的额外的比特个数，可以用来表示模型之间的差异。

KL Divergence of *q* to *p*: ****

KL Divergence of *p* to *q*: 



英文分词处理程序：<https://tartarus.org/martin/PorterStemmer/>

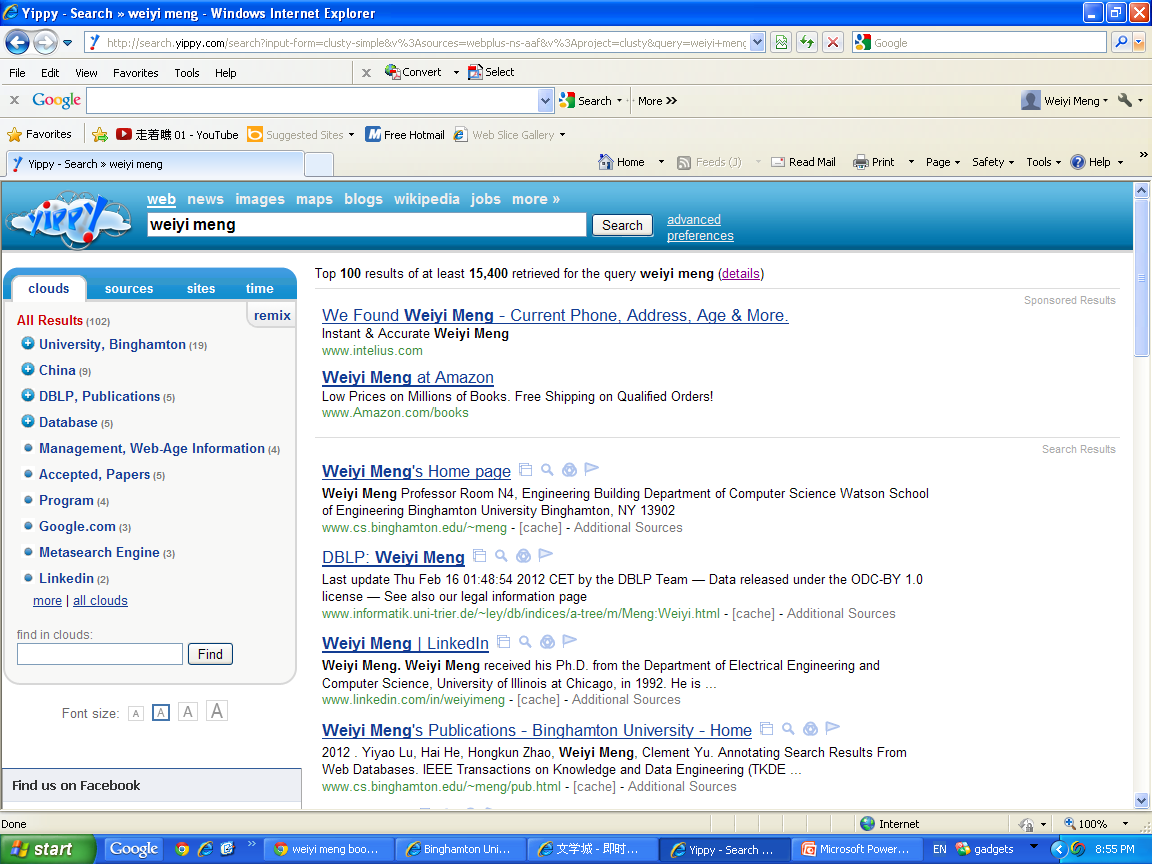
使用之前应当先进行对token的休整，比如说"pencils?"去除末尾的问号，否则会对stemmer的效果产生影响。C/C++版就三个函数给你进行使用，这三个函数定义前使用了extern标出，一个是create\_stemmer进行malloc结构体内存分配，还有stem传入stemmer结构体指针和字符串指针与大小返回给你这个分词认为字符串应该断在哪里，比如说pencils，返回一个5，\0应该在[5+1]。最后还有一个free\_stemmer来释放之前分配的结构体内存。

index处理程序：

open source text indexing packages as Lucene [2] and Lemur [20]

RISE indexing framework

老师曾说某一个多维度空间中两个点我们认为相距比较远，但是它们可能对于同一个点都是“相距较近”。



yippy是会将搜索结果自动归类的IR System，想到了川大购买的一个数据库，似乎也是有自动归类，点击归类以后就相当于是增加限制条件。