现代信息检索

Modern Information Retrieval

第9讲　完整搜索系统中的评分计算

Scores in a complete search system

*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 结果排序的动机

② 上一讲回顾

③ 结果排序的实现

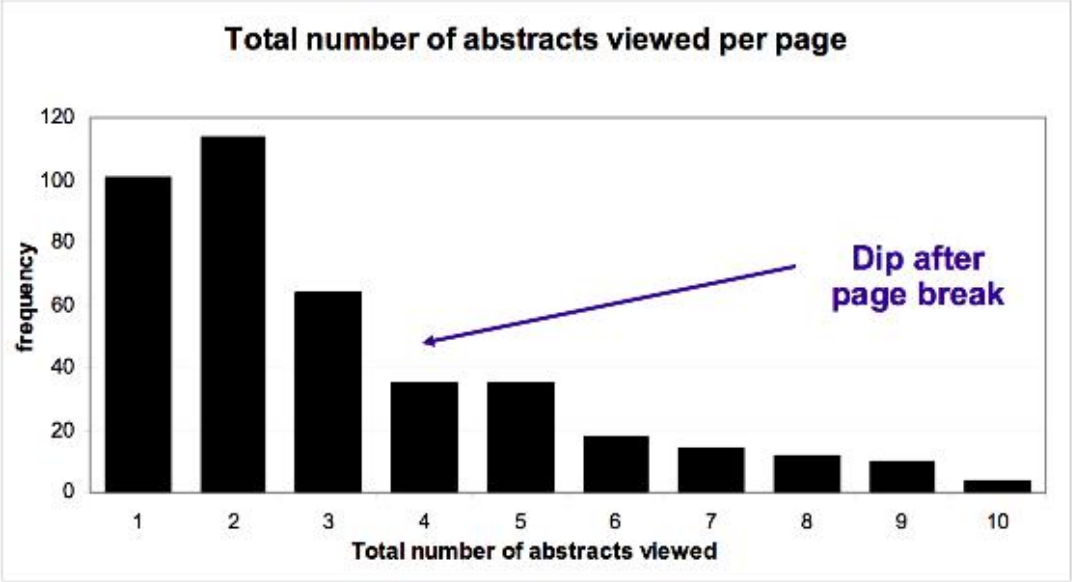
④ 完整的搜索系统

排序的重要性

- 不排序的问题严重性
  - 用户只希望看到一些而不是成千上万的结果
  - 很难构造只产生一些结果的查询
  - 即使是专家也很难
  - 排序能够将成千上万条结果缩减至几条结果，因此非常重要
- 接下来: 将介绍用户的相关行为数据
- 实际上，大部分用户只看1到3条结果
- 下面的讲义来自Dan Russell在JCDL会议上的讲话
- Dan Russell是Google的 “Über Tech Lead for Search Quality & User Happiness“

用户浏览的链接数

How many links do users view?

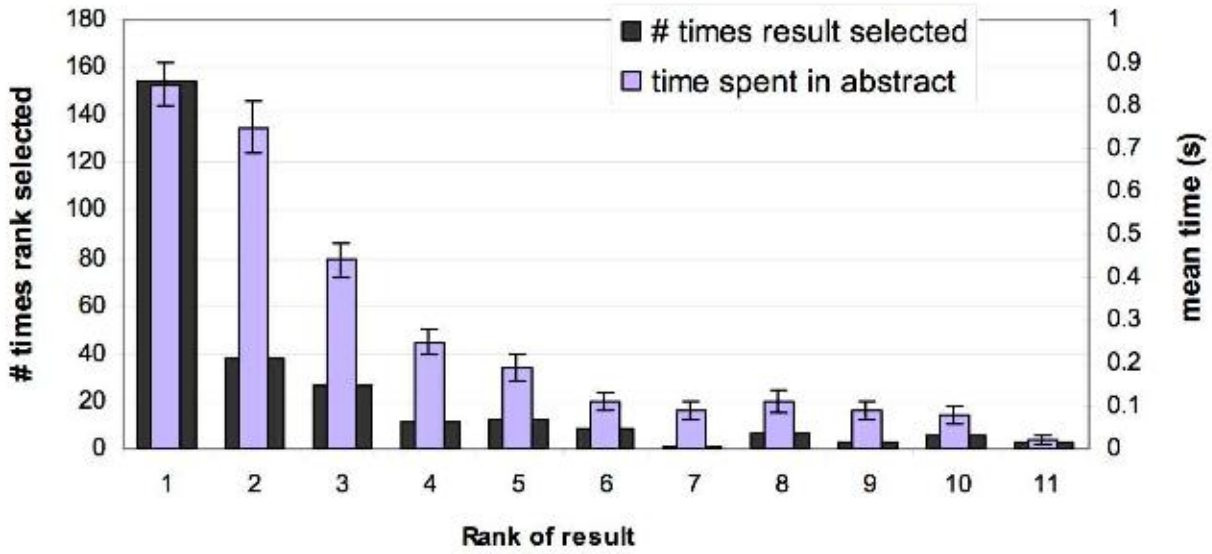


Mean: 3.07 Median/Mode: 2.00

Google

浏览 vs. 点击

Looking vs. Clicking



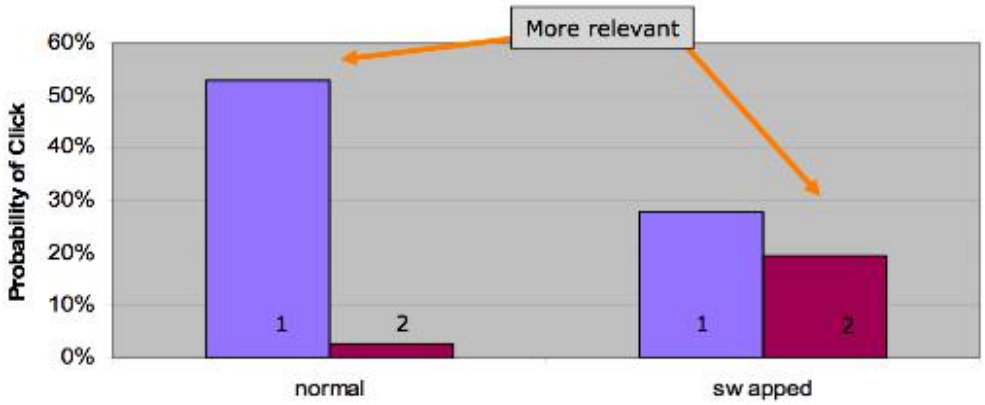
- Users view results one and two more often / thoroughly
- Users click most frequently on result one

Google

结果显示顺序对行为的影响

Presentation bias – reversed results

- Order of presentation influences where users look **AND** where they click



Google

排序的重要性: 小结

- 摘要阅读(Viewing abstracts): 用户更可能阅读第一页的结果的摘要
- 点击(Clicking): 点击的分布甚至更有偏向性
  - 一半情况下，用户点击排名最高的页面
  - 即使排名最高的页面不如排名第二的页面相关，仍然有接近30%的用户会点击它。
- → 正确排序相当重要
- → 排对最高的页面非常重要

提纲

① 结果排序的动机

② 上一讲回顾

③ 结果排序的实现

④ 完整的搜索系统

上一讲回顾

- 信息检索的评价方法
  - 不考虑序的评价方法(即基于集合)：P、R、F
  - 考虑序的评价方法：P/R曲线、MAP、NDCG
- 信息检索评测语料及会议
- 检索结果的摘要

正确率(Precision)和召回率(Recall)

- 正确率(Precision，简写为*P*) 是返回文档中真正相关的比率

$$\text{Precision} = \frac{\#(\text{relevant items retrieved})}{\#(\text{retrieved items})} = P(\text{relevant}|\text{retrieved})$$

- 召回率(Recall,*R*) 是返回结果中的相关文档占所有相关文档(包含返回的相关文档和未返回的相关文档)的比率

$$\text{Recall} = \frac{\#(\text{relevant items retrieved})}{\#(\text{relevant items})} = P(\text{retrieved}|\text{relevant})$$

正确率 vs. 召回率

	相关(relevant)	不相关(nonrelevant)
返回(retrieved)	真正例(true positives, *tp*)	伪正例(false positives, *fp*)
未返回(not retrieved)	伪反例(false negatives, *fn*)	真反例(true negatives, *tn*)

$$P = TP / (TP + FP)$$

$$R = TP / (TP + FN)$$

正确率和召回率相结合的指标：F值

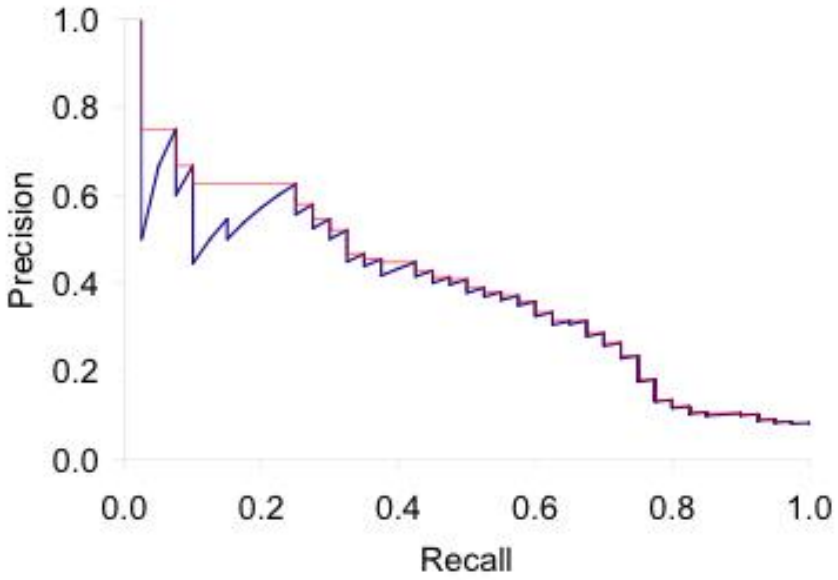
- F 允许正确率和召回率的折中

$$F = \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R}} = \frac{(\beta^2+1)PR}{\beta^2 P + R} \quad \text{where } \beta^2 = \frac{1-\alpha}{\alpha}$$

- $\alpha \in [0, 1]$ ， $\beta^2 \in [0,\infty]$
- 常用参数: balanced F ， $\beta = 1$ or $\alpha = 0.5$
  - 实际上是正确率和召回率的调和平均数（harmonic mean）

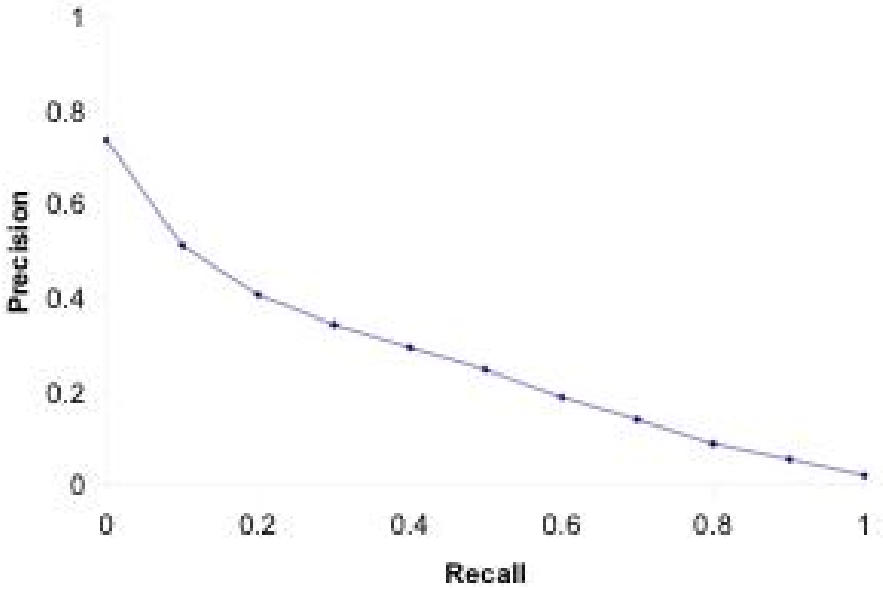
$$\frac{1}{F} = \frac{1}{2}\left(\frac{1}{P} + \frac{1}{R}\right)$$

正确率-召回率曲线



- 每个点对应top k上的结果 (k = 1, 2, 3, 4, . . .).
- 插值 (红色): 将来所有点上的最高结果
- 插值的原理：如果正确率和召回率都升高，那么用户可能愿意浏览更多的结果

平均的 11-点正确率/召回率曲线



- 计算每个召回率点(0.0, 0.1, 0.2, . . .)上的插值正确率
- 对每个查询都计算一遍
- 在查询上求平均
- 该曲线也是ＴＲＥＣ评测上常用的指标之一

MAP

- 平均正确率(Average Precision, AP)：对不同召回率点上的正确率进行平均
  - 未插值的AP：某个查询Q共有6个相关结果，某系统排序返回了5篇相关文档，其位置分别是第1，第2，第5，第10，第20位，则
    AP=(1/1+2/2+3/5+4/10+5/20+0)/6
- 多个查询的AP的平均值称为系统的MAP(Mean AP)
- MAP是IR领域使用最广泛的指标之一

NDCG

- BV(Best Vector)：假定*m*个3，*l*个2，*k*个1，其他都是0

$$\mathrm{BV}[i] = \begin{cases} 3, & \text{if } i \le m, \\ 2, & \text{if } m < i \le m+l, \\ 1, & \text{if } m+l < i \le m+l+k, \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

$$\mathrm{I}' = \langle 3, 3, 3, 2, 2, 2, 1, 1, 1, 1, 0, 0, 0, \ldots \rangle.$$

$$\mathrm{CG}'_{\mathrm{I}} = \langle 3, 6, 9, 11, 13, 15, 16, 17, 18, 19, 19, 19, 19, \ldots \rangle$$
$$\mathrm{DCG}'_{\mathrm{I}} = \langle 3, 6, 7.89, 8.89, 9.75, 10.52, 10.88, 11.21, 11.53, 11.83, 11.83, 11.83, \ldots \rangle.$$

NDCG

- Normalized (D)CG

$$norm\text{-}vect(\mathrm{V},\mathrm{I}) = \langle \mathrm{v}_1/\mathrm{i}_1, \mathrm{v}_2/\mathrm{i}_2, \ldots, \mathrm{v}_\mathrm{k}/\mathrm{i}_\mathrm{k} \rangle. \tag{5}$$

$$\begin{aligned} \mathrm{nCG}' &= norm\text{-}vect(\mathrm{CG}', \mathrm{CG}'_\mathrm{I}) \\ &= \langle 1, 0.83, 0.89, 0.73, 0.62, 0.6, 0.69, 0.76, 0.89, 0.84, \ldots \rangle. \end{aligned}$$

另一种NDCG的计算方法

- 加大相关度本身的权重，原来是线性变化，现在是指数变化，相关度3、2、1　在计算时用2^3、2^2、2^1

$$\mathrm{NDCG}(Q,k)=\frac{1}{|Q|}\sum_{j=1}^{|Q|}Z_{j,k}\sum_{m=1}^{k}\frac{2^{R(j,m)}-1}{\log(1+m)} \qquad (8\text{-}9)$$

- 据说搜索引擎公司常用这个公式

标准的评价会议: TREC

- TREC = Text Retrieval Conference (TREC)
- 美国标准技术研究所 (NIST)组织
- TREC实际上包含了对多个任务的评测
- 最出名的任务: TREC Ad Hoc任务，1992到1999年前8届会议中的标准任务

- TREC disk包含189百万篇文档，主要是新闻报道，有 450个信息需求
- 由于人工标注的代价太大，所有没有完整的相关性判定
- 然而，NIST采用了一种所谓结果缓冲(pooling)的办法来进行人工标注，首先将所有参测系统的前k个结果放到一个缓冲池(pool)，然后仅对缓冲池的文档进行标注，并认为所有的相关文档均来自该缓冲池中。

提纲

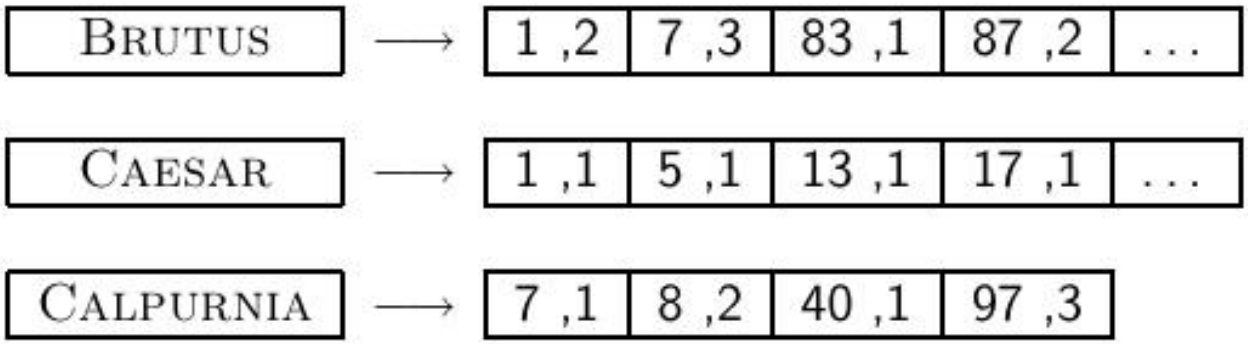
① 上一讲回顾

② 结果排序的动机

③ 结果排序的实现

④ 完整的搜索系统

将词项频率tf存入倒排索引中



- 当然也需要位置信息，上面没显示出来

倒排索引中的词项频率存储

- 每条倒排记录中，除了docIDd 还要存储$tf_{t,d}$
- 通常存储是原始的整数词频，而不是对数词频对应的实数值
- 这是因为实数值不易压缩
- 对tf采用一元码编码效率很高
- 总体而言，额外存储tf所需要的开销不是很大：采用位编码压缩方式，每条倒排记录增加不到一个字节的存储量
- 或者在可变字节码方式下每条倒排记录额外需要一个字节即可

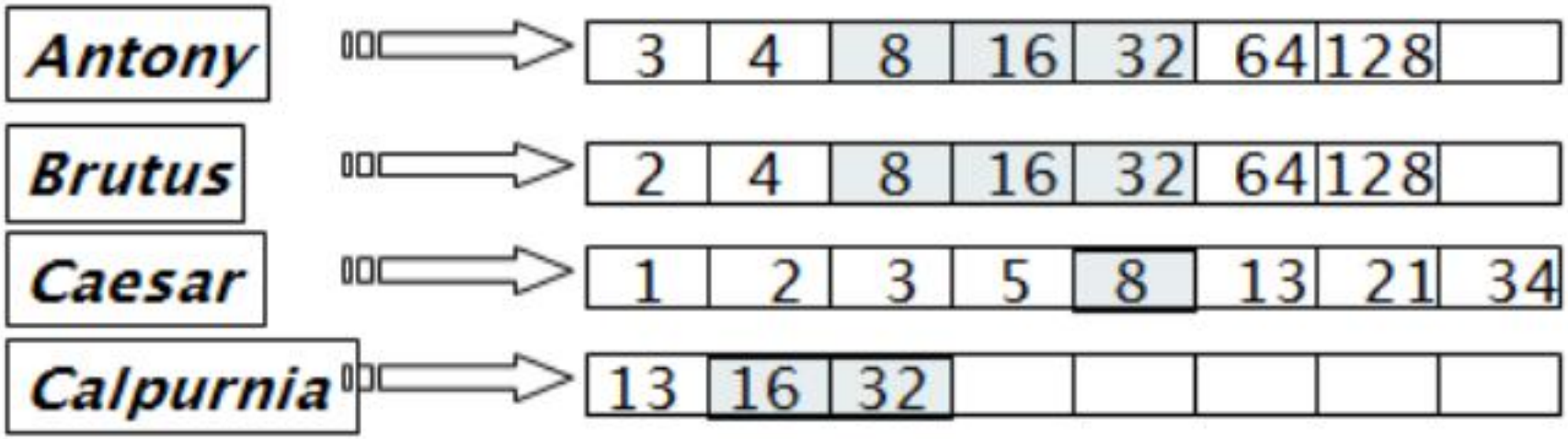
两种常见的评分累加算法

- 以词项为单位(term-at-a-time, TAAT)，首先获得词项t的posting list，然后累加得分
- 另一种常见算法DAAT使用的是以文档为单位的计算，首先获得包含查询词的所有文档，将这些文档按照静态评分排序，然后依次累加得分

余弦相似度的TAAT计算算法

```
CosineScore(q)
 1  float Scores[N] = 0
 2  float Length[N]
 3  for each query term t
 4  do calculate w_{t,q} and fetch postings list for t
 5      for each pair(d, tf_{t,d}) in postings list
 6      do Scores[d] += w_{t,d} × w_{t,q}
 7  Read the array Length
 8  for each d
 9  do Scores[d] = Scores[d]/Length[d]
10  return Top K components of Scores[]
```

从倒排索引取出倒排记录表，依次计算权重

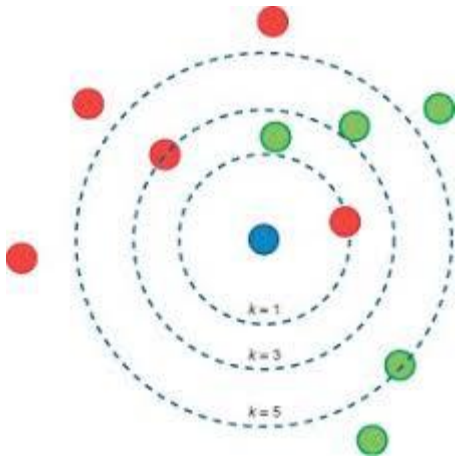


精确top *K*检索及其加速办法

- 目标：从文档集的所有文档中找出K 个离查询最近的文档
- (一般)步骤：对每个文档评分(余弦相似度)，按照评分高低排序，选出前K个结果
- 如何加速：
  - 思路一：加快每个余弦相似度的计算
  - 思路二：不对所有文档的评分结果排序而直接选出Top K篇文档
  - 思路三：能否不需要计算所有N篇文档的得分？

精确top K检索加速方法一：快速计算余弦

- 检索排序就是找查询的K近邻
- 一般而言，在高维空间下，计算余弦相似度没有很高效的方法
- 但是如果查询很短，是有一定办法加速计算的，而且普通的索引能够支持这种快速计算



特例– 不考虑查询词项的权重

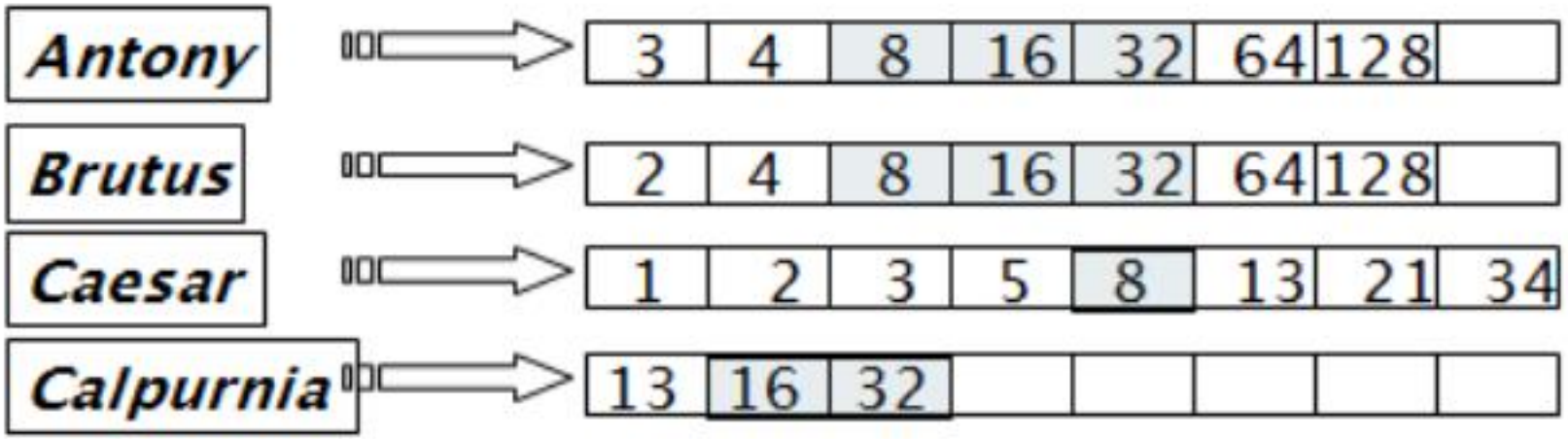
- 查询词项无权重
  - 相当于假设每个查询词项都出现1次
- 于是，不需要对查询向量进行归一化
  - 于是可以对上一讲给出的余弦相似度计算算法进行轻微的简化

快速余弦相似度计算: 无权重查询

```
FastCosineScore(q)
 1  float Scores[N] = 0
 2  for each d
 3  do Initialize Length[d] to the length of doc d
 4  for each query term t
 5  do calculate w_{t,q} and fetch postings list for t
 6      for each pair(d, tf_{t,d}) in postings list
 7      do add wf_{t,d} to Scores[d]
 8  Read the array Length[d]
 9  for each d
10  do Divide Scores[d] by Length[d]
11  return Top K components of Scores[]
```

Figure 7.1 A faster algorithm for vector space scores.

前面的例子

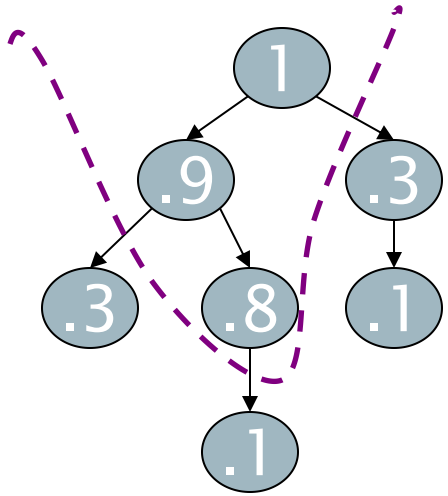

Antony
3 4 8 16 32 64 128
Brutus
2 4 8 16 32 64 128
Caesar
1 2 3 5 8 13 21 34
Calpurnia
13 16 32

精确top k检索加速方法二：堆法N中选K

- 检索时，通常只需要返回前K条结果
  - 可以对所有的文档评分后排序，选出前K个结果，但是这个排序过程可以避免
- 令 J = 具有非零余弦相似度值的文档数目
  - 从J中选K个最大的
- 不对所有文档进行排序，只需要挑出最高的K个结果

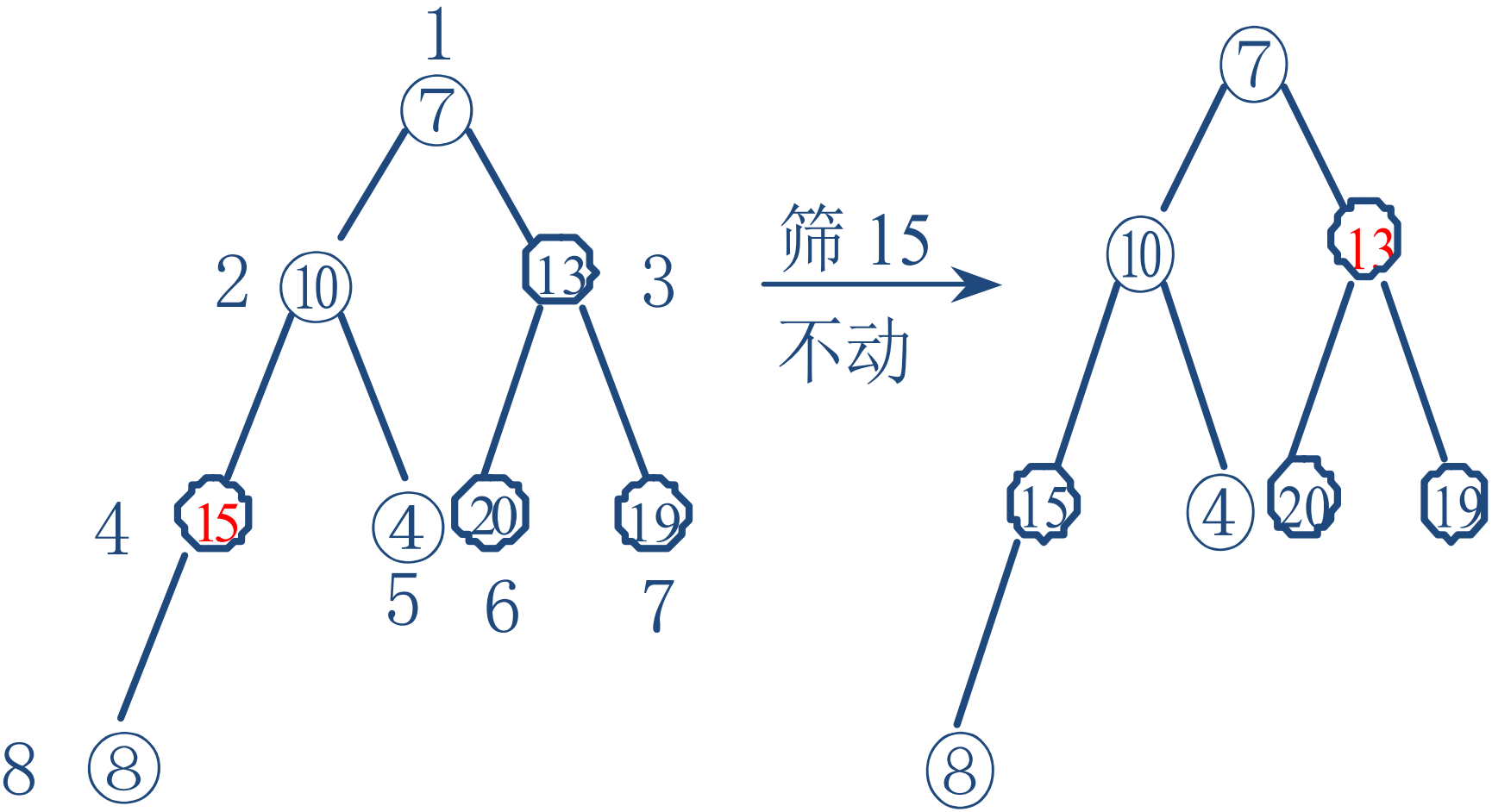
堆方法

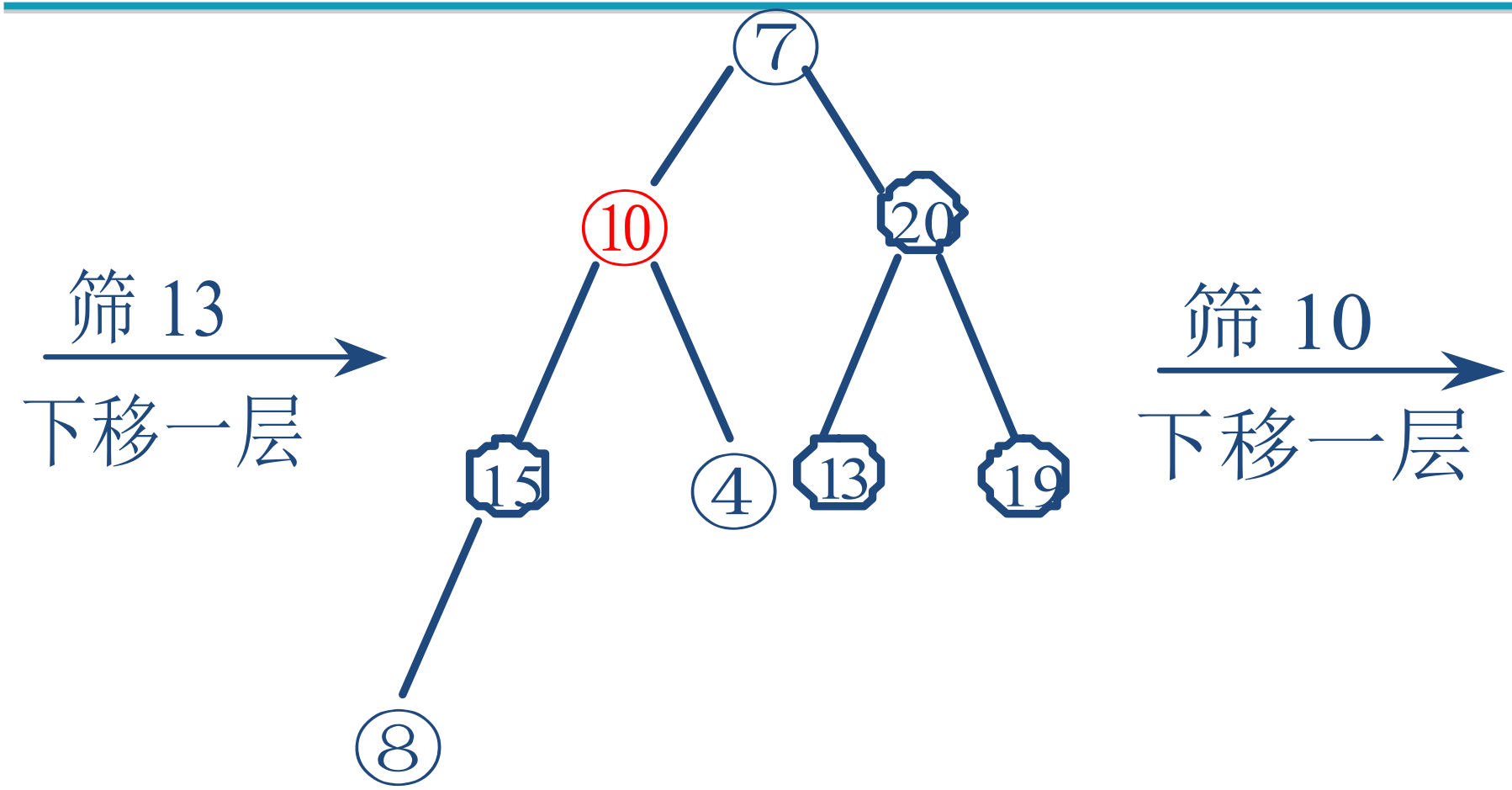
- 堆：二叉树的一种，每个节点上的值> 子节点上的值 (Max Heap)
  - 构建最大堆，然后选择前K个节点
- 堆构建：需要 2J 次操作
- 选出前K个结果：每个结果需要2log J 步
- 如果 J=1M, K=100, 那么代价大概是全部排序代价的10%

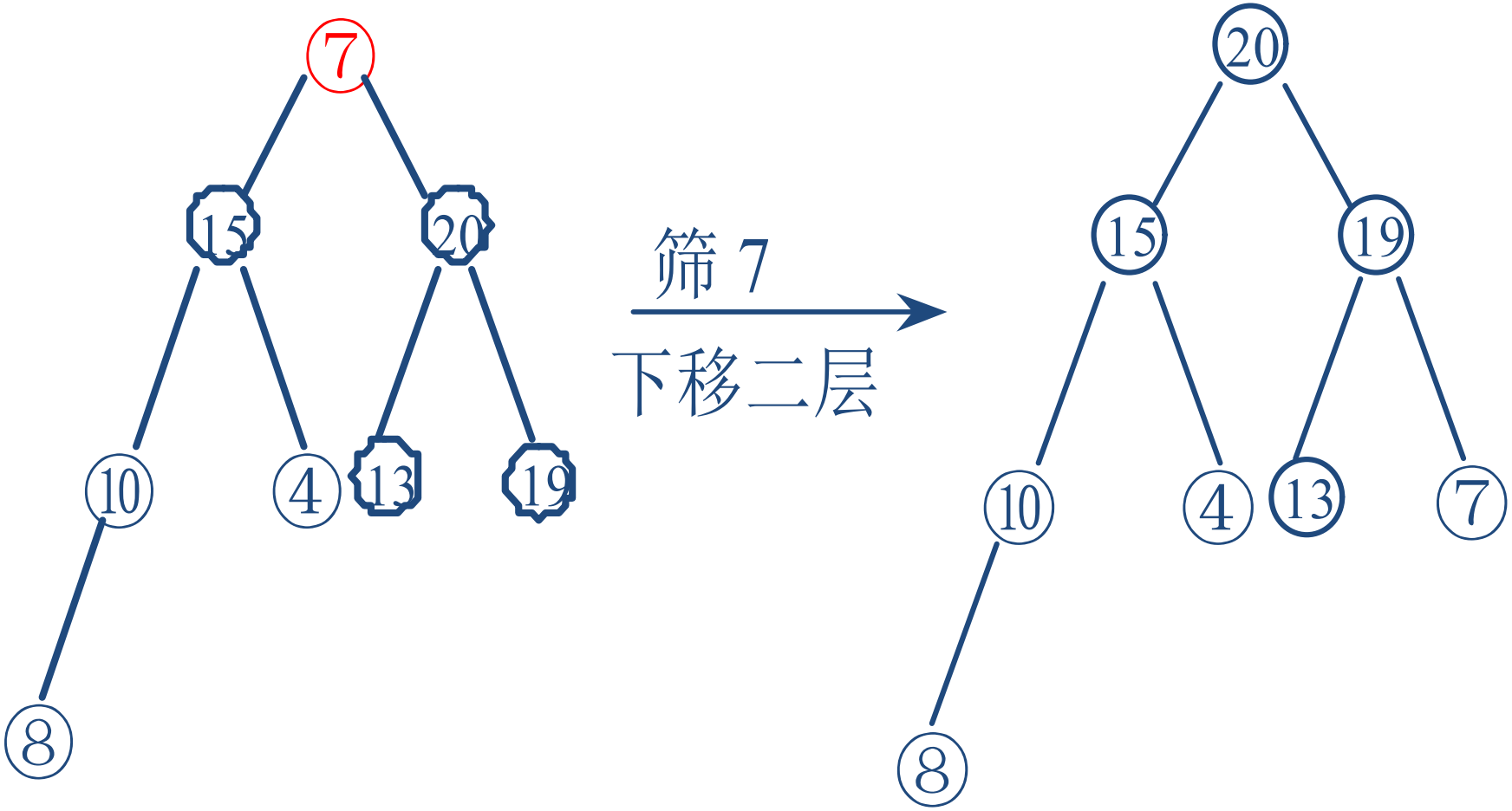


(最大)堆构建样例(筛选shift法-摘自网上课件)

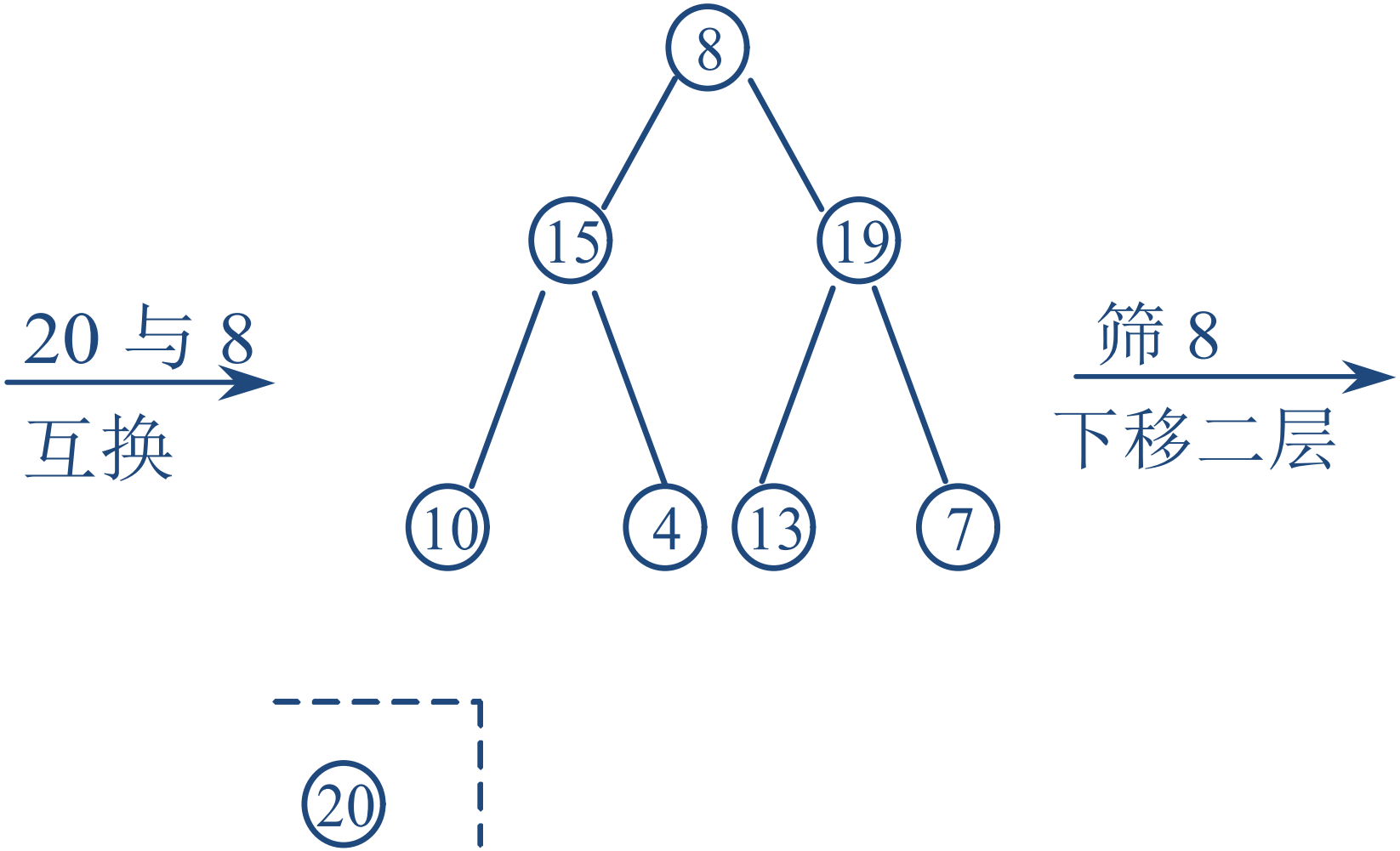
- 7，10，13，15，4，20，19，8（数据个数n=8）。

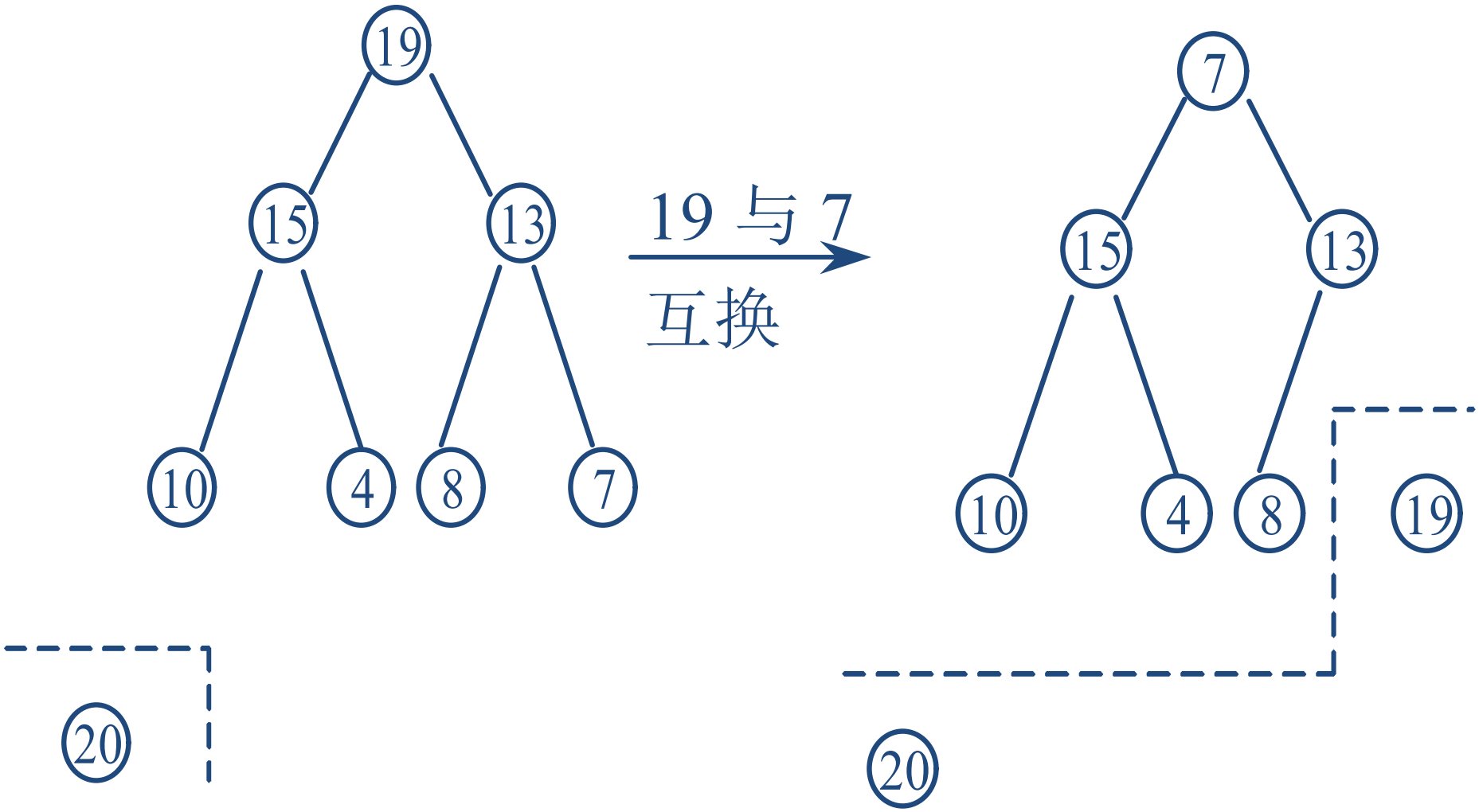


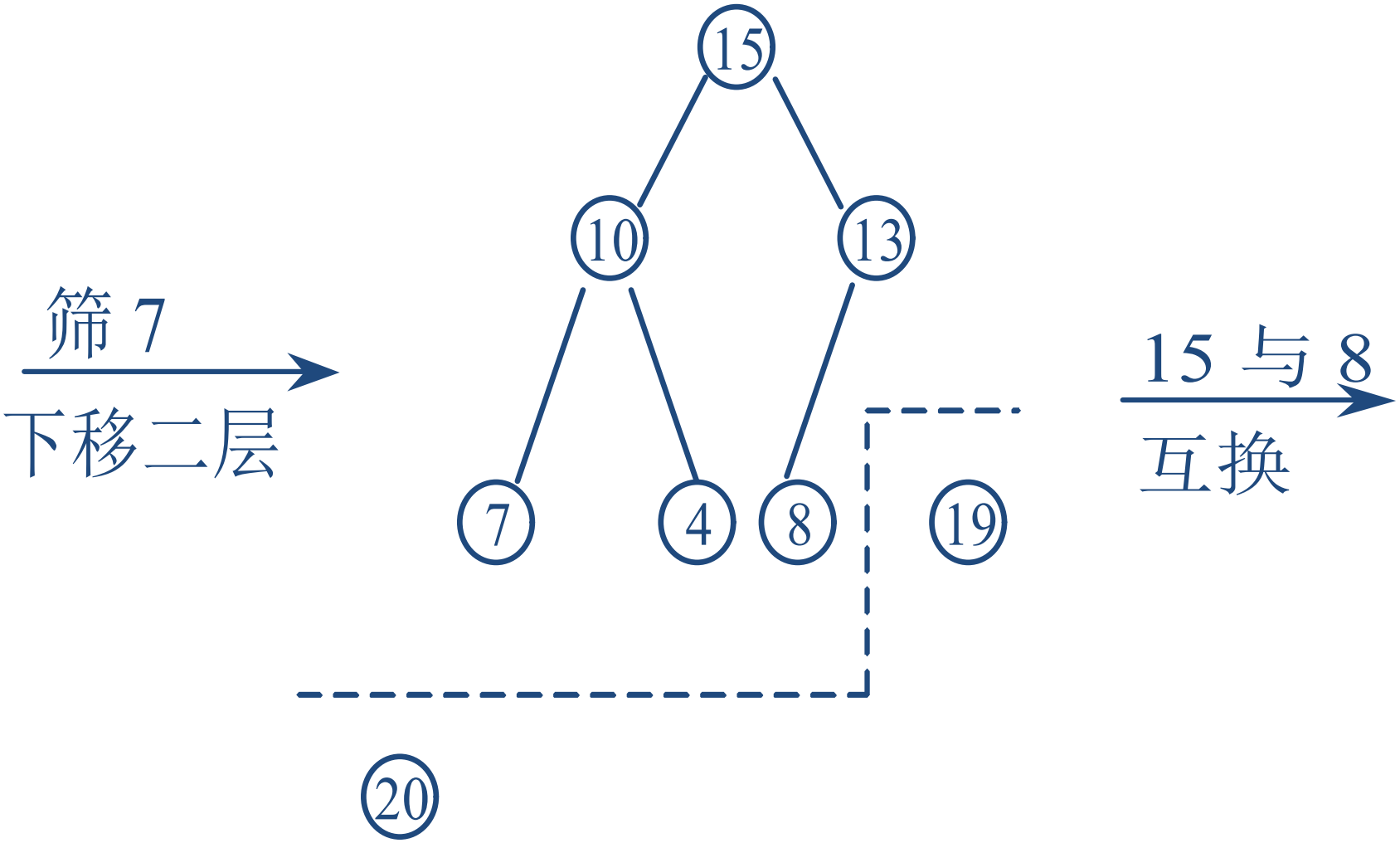

筛 13
下移一层
7
10
20
15
4
13
19
8
筛 10
下移一层

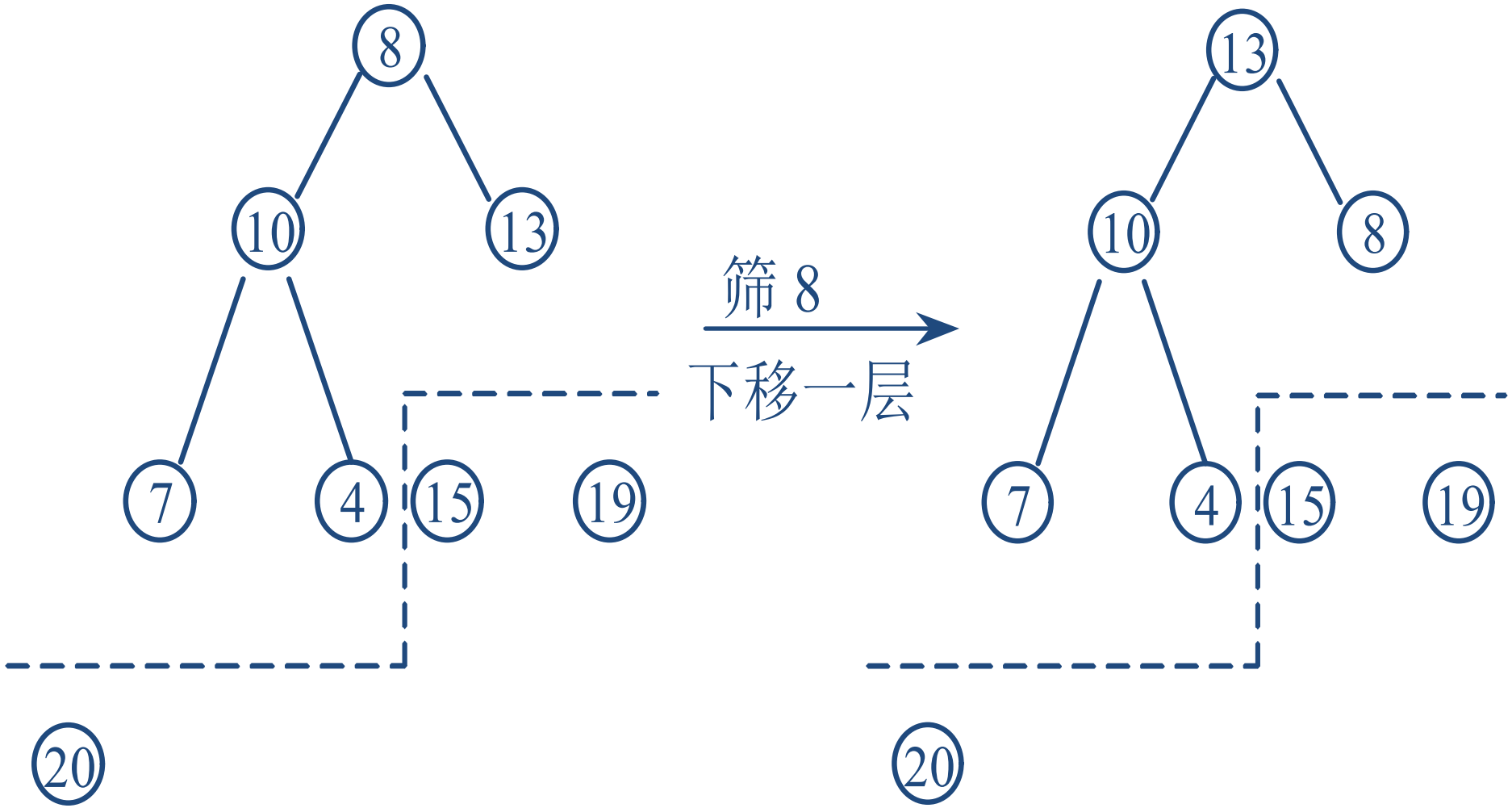

7
15
20
10
4
13
19
8
筛 7
下移二层
20
15
19
10
4
13
7
8

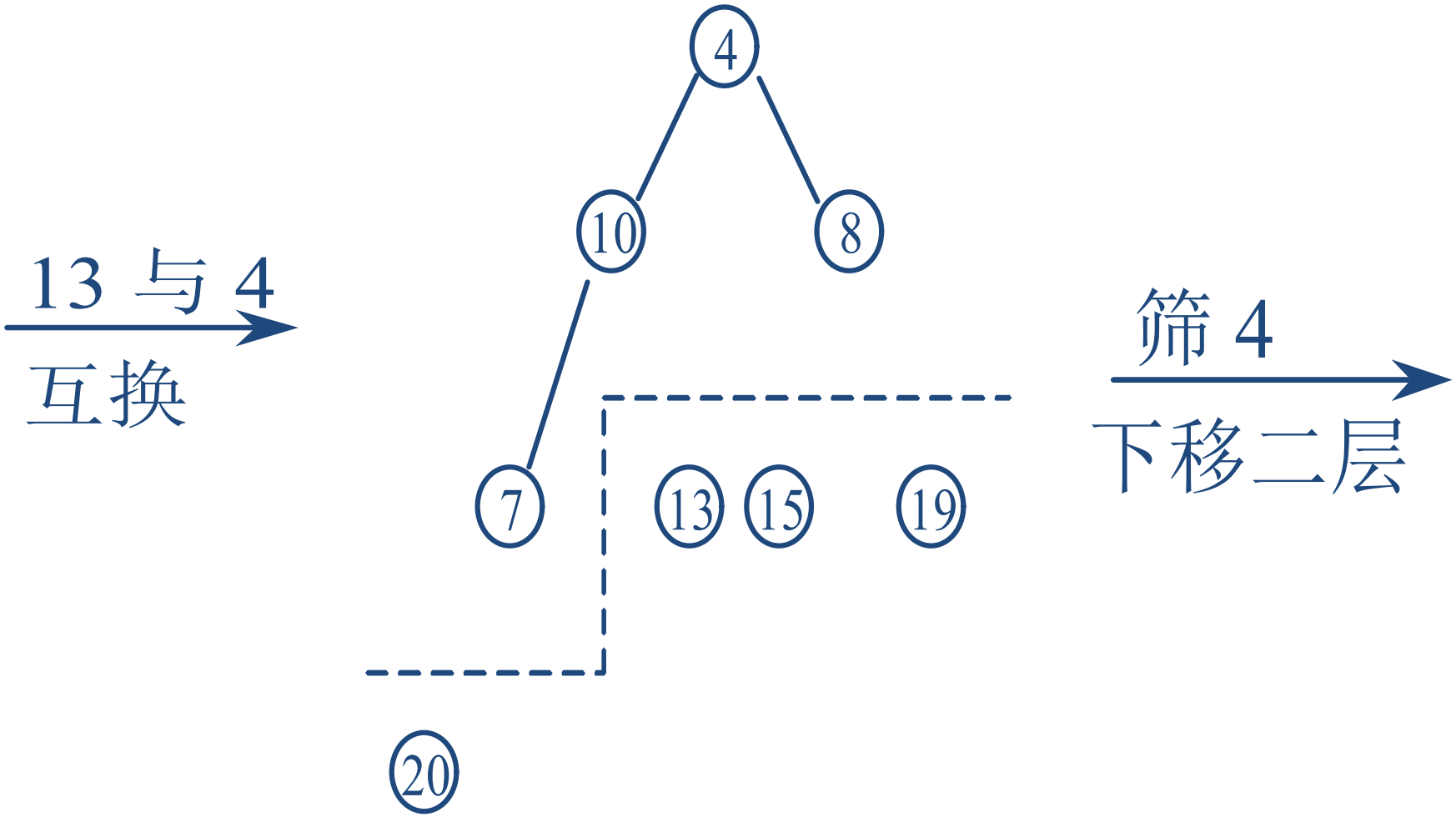
利用堆选出Top K (=4)




19
15
13
10
4
8
7
20
19 与 7
互换
7
15
13
10
4
8
19
20


筛 7
下移二层
15
10
13
7
4
8
19
20
15 与 8
互换


8
10
13
7
4
15
19
20
筛 8
下移一层
13
10
8
7
4
15
19
20


4
10
8
7
13
15
19
20
13 与 4
互换
筛 4
下移二层

精确top K检索加速方法三：提前终止计算

- 到目前为止的倒排记录表都按照docID排序
- 接下来将采用与查询无关的另外一种反映结果好坏程度的指标(静态质量)
  - 例如: 页面d的PageRank g(d), 就是度量有多少好页面指向d的一种指标 (参考第 21章)
  - 于是可以将文档按照PageRank排序

    $$g(d1) > g(d2) > g(d3) > \ldots$$

  - 将PageRank和余弦相似度线性组合得到文档的最后得分
- $$\text{net-score}(q, d) = g(d) + \cos(q, d)$$

提前终止计算

- 假设:
  - (i) g → [0, 1];
  - (ii) 检索算法按照d1,d2,…，依次计算(为文档为单位的计算，document-at-a-time)，当前处理的文档的 g(d) < 0.1；
  - (iii) 而目前找到的top K 的得分中最小的都 > 1.2
- 由于后续文档的得分不可能超过1.1 (cos(q,d) <1)
- 所以，我们已经得到了top K结果，不需要再进行后续计算

精确top K检索的问题

- 仍然无法避免大量文档参与计算
- 一个自然而然的问题就是能否尽量减少参与计算文档数目，即使不能完全保证正确性也在所不惜。
  - 即采用这种方法得到的top K虽然接近但是并非真正的top K----非精确top K检索

非精确top K检索的可行性

- 检索是为了得到与查询匹配的结果，该结果要让用户满意
- 余弦相似度是刻画用户满意度的一种方法
- 非精确top K的结果如果和精确top K的结果相似度相差不大，应该也能让用户满意

一般思路

- 找一个文档集合A，K<|A|<<N，利用A中的top K结果代替整个文档集的top K结果
  - 即给定查询后，A是整个文档集上近似剪枝得到的结果
  - 上述思路不仅适用于余弦相似度得分，也适用于其他相似度计算方法

方法一：索引去除(Index elimination)

- 一般检索方法中，通常只考虑至少包含一个查询词项的文档
- 可以进一步拓展这种思路
  - 只考虑那些包含高idf查询词项的文档
  - 只考虑那些包含多个查询词项的文档(比如达到一定比例，3个词项至少出现2个，4个中至少出现3个等等)

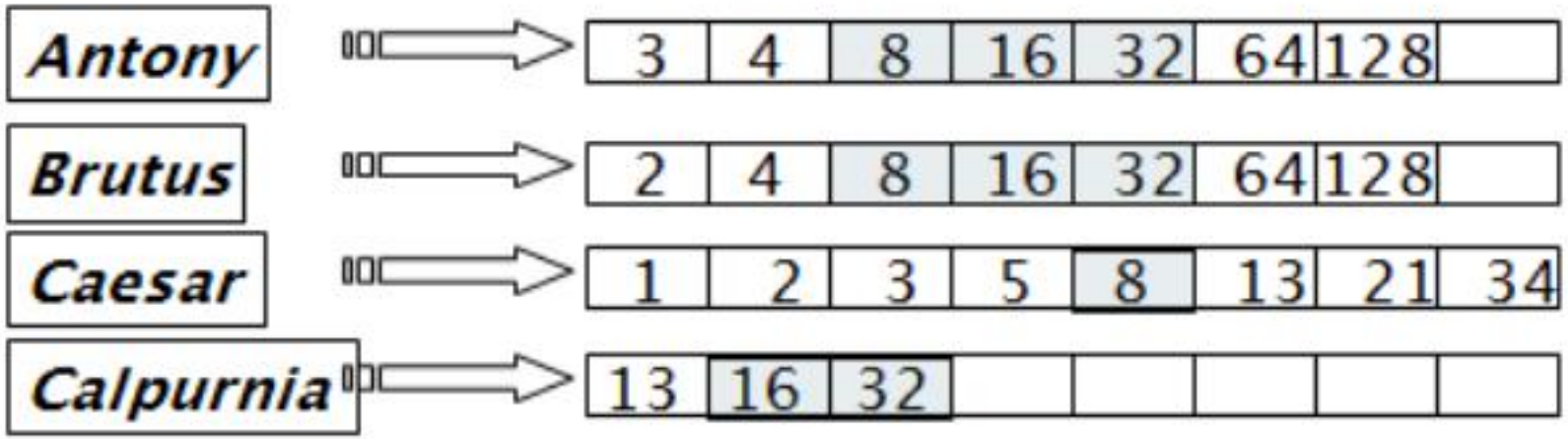
仅考虑高idf词项

- 对于查询 catcher in the rye
- 仅考虑包含catcher和rye的文档的得分
- 直觉： 文档当中的in 和 the不会显著改变得分因此也不会改变得分顺序
- 优点:
  - 低idf词项会对应很多文档，这些文档会排除在集合A之外

仅考虑包含多个词项的文档

- Top K的文档至少包含一个查询词项
- 对于多词项查询而言，只需要计算包含其中大部分词项的文档
  - 比如，至少4中含3
  - 这相当于赋予了一种所谓软合取(soft conjunction)的语义 (早期Google使用了这种语义)
- 这种方法很容易在倒排记录表合并算法中实现

包含4个查询词项中的3个



仅对文档8、16和 32进行计算

方法二：胜者表(Champion list)

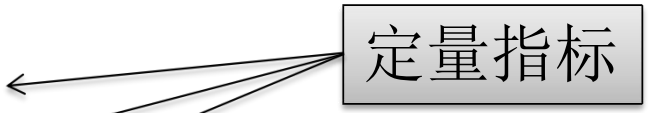
- 对每个词项t，预先计算出其倒排记录表中权重最高的r篇文档，如果采用tfidf机制，即tf最高的r篇
  - 这r篇文档称为t的胜者表
  - 也称为优胜表(fancy list)或高分文档(top docs)
- 注意：r 比如在索引建立时就已经设定
  - 因此，有可能 r < K
- 检索时，仅计算某些词项的胜者表中包含的文档集合的并集
  - 从这个集合中选出top K作为最终的top K

课堂思考

- 胜者表方式和前面的索引去除方式有什么关联？如何融合它们？
- 如何在一个倒排索引当中实现胜者表？
  - 提醒：胜者表与docID大小无关

方法三：静态质量得分排序方式

- 我们希望排名靠前的文档不仅相关度高(relevant)，而且权威度也大(authoritative)
- 相关度常常采用余弦相似度得分来衡量
- 而权威度往往是一个与查询无关的量，是文档本身的属性
- 权威度示例
  - Wikipedia在所有网站上的重要性
  - 某些权威报纸上的文章
  - 论文的引用量
  - 被 diggs, Y!buzzes或del.icio.us等网站的标注量
  - Pagerank（前面介绍精确top K检索时也提及）



权威度计算

- 为每篇文档赋予一个与查询无关的(query-independent) [0,1]之间的值，记为g(d)，例如Pagerank
- 同前面一样，最终文档排名基于g(d)和相关度的线性组合。
  - net-score(q,d) = g(d) + cosine(q,d)
  - 可以采用等权重，也可以采用不同权重
  - 可以采用任何形式的函数，而不只是线性函数
- 接下来我们的目标是找net-score最高的top K文档（非精确检索）

基于net-score的Top K文档检索

- 首先按照g(d)从高到低将倒排记录表进行排序
- 该排序对所有倒排记录表都是一致的(只与文档本身有关)
- 因此，可以并行遍历不同查询词项的倒排记录表来
  - 进行倒排记录表的合并
  - 及余弦相似度的计算
- 课堂练习：写一段伪代码来实现上述方式下的余弦相似度计算

一种典型的Document-at-a-time (DAAT)流程

```
D=mergeAllPostings(postings(Q));
sortDecreasingByGd(D);
For p in postings(Q);
    sortDecreasingByGd(p);
For d in D;do
    if g(d)< threshold // 提前结束条件，可以换成其它条件，例如处理时间
          end process;
for p in postings;
   if d in p;{
       score+=w(t, d);
   }
   score+=g(d)
done;
```

利用g(d)排序的优点

- 这种排序下，高分文档更可能在倒排记录表遍历的前期出现

- 在时间受限的应用当中 (比如，任意搜索需要在50ms内返回结果), 上述方式可以提前结束倒排记录表的遍历

将g(d)排序和胜者表相结合

- 对每个词项维护一张胜者表，该表中放置了r篇g(d) + $tf\text{-}idf_{t,d}$ 值最高的文档

- 检索时只对胜者表进行处理

高端表(High list)和低端表(Low list)

- 对每个词项，维护两个倒排记录表，分别称为高端表和低端表
  - 比如可以将高端表看成胜者表
- 遍历倒排记录表时，仅仅先遍历高端表
  - 如果返回结果数目超过K，那么直接选择前K篇文档返回
  - 否则，继续遍历低端表，从中补足剩下的文档数目
- 上述思路可以直接基于词项权重，不需要全局量g(d)
- 实际上，相当于将整个索引分层

方法四：影响度(Impact)排序

- 如果只想对 $wf_{t,d}$ 足够高的文档进行计算
- 那么就可以将文档按照 $wf_{t,d}$ 排序
- 需要注意的是：这种做法下，倒排记录表的排序并不是一致的(排序指标和查询相关)

- 那么如何实现top K的检索?
  - 以下介绍两种做法

1. 提前结束法

- 遍历倒排记录表时，可以在如下情况之一发生时停止：
  - 遍历了固定的文档数目r
  - $wf_{t,d}$ 低于某个预定的阈值
- 将每个词项的结果集合合并

- 仅计算合并集合中文档的得分

2. 将词项按照idf排序

- 对于多词项组成的查询，按照idf从大到小扫描词项
- 在此过程中，会不断更新文档的得分(即本词项的贡献)，如果文档得分基本不变的话，停止
- 可以应用于余弦相似度或者其他组合得分

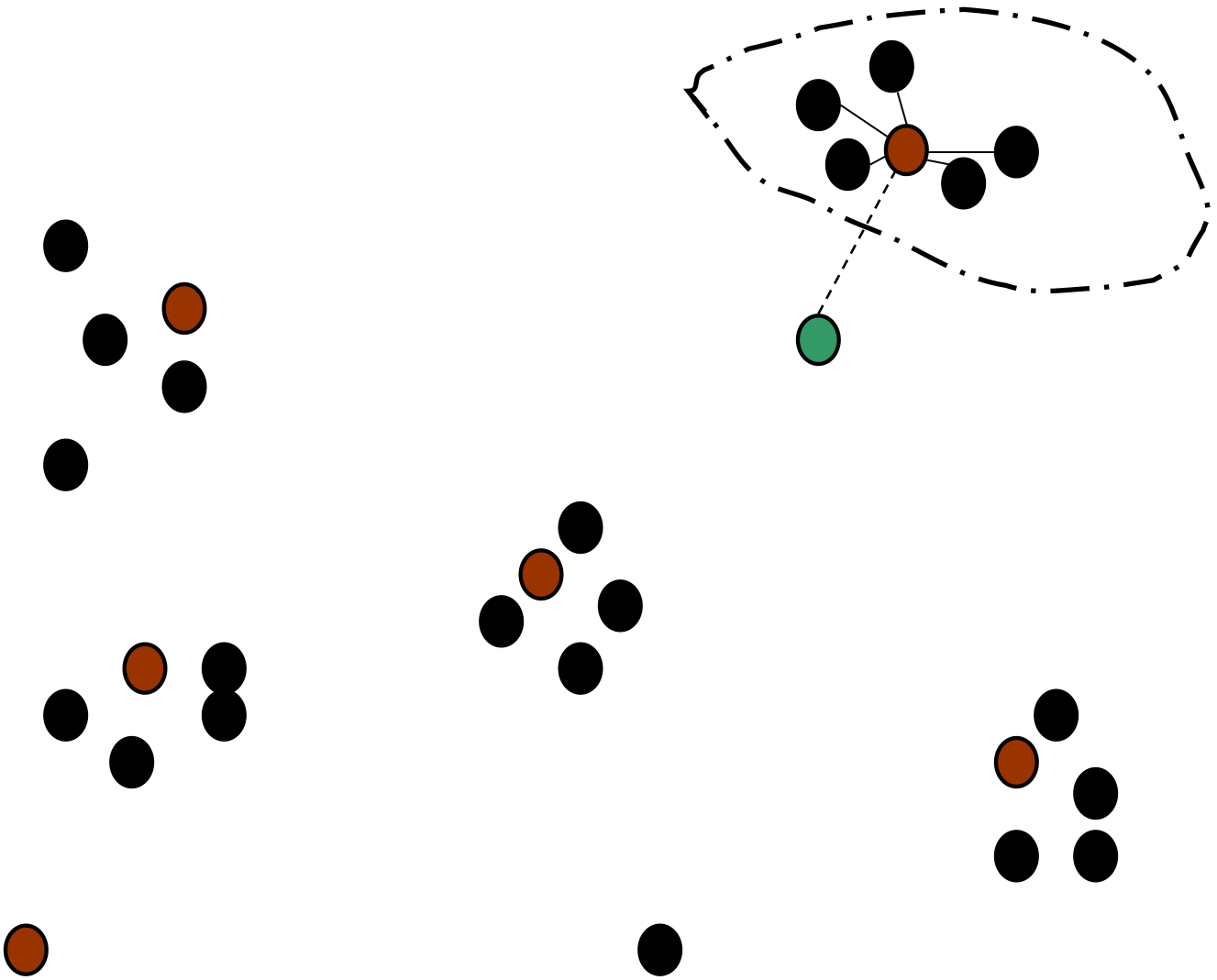
方法五： 簇剪枝(Cluster pruning)

- 一种基于聚类的方法
- 随机选 √N 篇文档作为先导者
- 对于其他文档，计算和它最近的先导者
  - 这些文档依附在先导者上面，称为追随者(follower)
  - 这样一个先导者平均大约有 ~ √N 个追随者

查询处理过程

- 给定查询 Q, 找离它最近的先导者L
- 从L及其追随者集合中找到前K个与Q最接近的文档返回

可视化示意图



为什么采用随机抽样？

- 速度快
- 先导者能够反映数据的分布情况

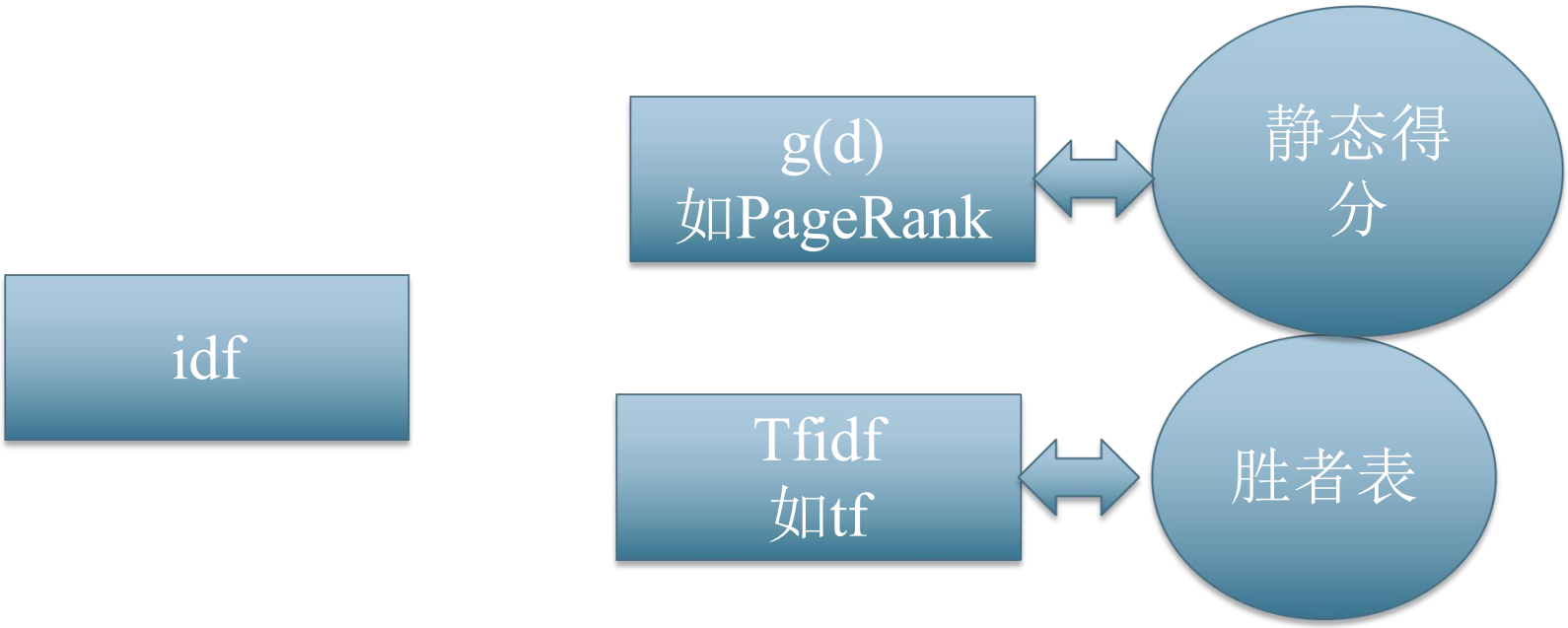
一般化变形

- 每个追随者可以附着在b1 (比如3)个最近的先导者上
- 对于查询，可以寻找最近的b2 (比如4)个先导者及其追随者

课堂练习

- 为了找到最近的先导者，需要计算多少次余弦相似度？
  - 为什么第一步中采用 √N 个先导者？
- 上一张讲义中的常数 b1, b2 会对结果有什么影响？
- 设计一个例子，上述方法可能会失败，比如返回的K篇文档中少了一篇真正的top K文档。
  - 这在随机抽样下是有可能的。

小结



非docID的倒排记录表排序方法（1）

- 到目前为止：倒排记录表都按照docID排序
- 另外的一种方法：与查询无关的一种反映结果好坏程度的指标
- 例如: 页面d的PageRank $g(d)$, 就是度量有多少好页面指向d的一种指标 (chapter 21)
- 将文档按照PageRank排序 $g(d_1) > g(d_2) > g(d_3) > \ldots$
- 计算文档的某个组合得分

$$\text{net-score}(q, d) = g(d) + \cos(q, d)$$

- 在这种机制下，能够在扫描倒排记录表时提前结束计算

以文档为单位(Document-at-a-time)的处理理

- 按照docID排序和按照PageRank排序都与词项本身无关(即两者都是文档的固有属性)，因此在全局这种序都是一致的。
- 上述计算余弦相似度的方法可以采用以文档为单位的处理方式。
- 即在开始计算文档d_{i+1}的得分之前，先得到文档d_i的得分。

- 另一种方式: 以词项为单位(term-at-a-time)的处理

以词项为单位(Term-at-a-time)的处理方式

- 最简单的情况：对第一个查询词项，对它的倒排记录表进行完整处理
- 对每个碰到的docID设立一个累加器
- 然后，对第二个查询词项的倒排记录表进行完整处理
- . . . 如此循环往复

以词项为单位(Term-at-a-time)的处理算法

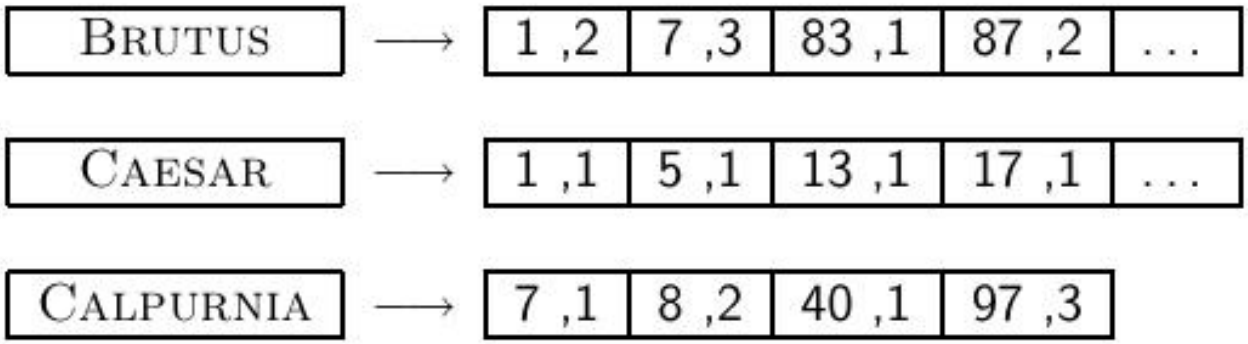
```
CosineScore(q)
 1  float Scores[N] = 0
 2  float Length[N]
 3  for each query term t
 4  do calculate w_{t,q} and fetch postings list for t
 5      for each pair(d, tf_{t,d}) in postings list
 6      do Scores[d] += w_{t,d} × w_{t,q}
 7  Read the array Length
 8  for each d
 9  do Scores[d] = Scores[d]/Length[d]
10  return Top k components of Scores[]
```

The elements of the array "Scores" are called accumulators.

余弦得分的计算

- 对于Web来说(200亿页面), 在内存中放置包含所有页面的累加器数组是不可能的
- 因此，仅对那些出现在查询词项倒排记录表中的文档建立累加器
- 这相当于，对那些得分为0的文档不设定累加器(即那些不包含任何查询词项的文档)

累加器举例



- 查询: [Brutus Caesar]:
- 仅为文档 1, 5, 7, 13, 17, 83, 87设立累加器
- 不为文档 8, 40, 85 设立累加器

瓶颈的消除

- 可以使用前面讨论的堆/优先队列结构
- 可以进一步将文档限制在那些在包含高idf值的非零得分文档
- 或者强制执行一个与查询 (类似Google): 在每个查询词项上都要得到非零余弦相似度值
- 例子: 为[Brutus Caesar]仅建立一个累加器
- 这是因为仅有d_1 同时包含这两个词

WAND（Weak AND） 评分算法

- DAAT 评分算法的一种
- 基本思想 - 分支和界限
  - 实时维护一个阈值 – 例如，当前第K高的评分
  - 去除cosine评分肯定低于阈值的文档
  - 仅为未去除的文档计算精确cosine评分

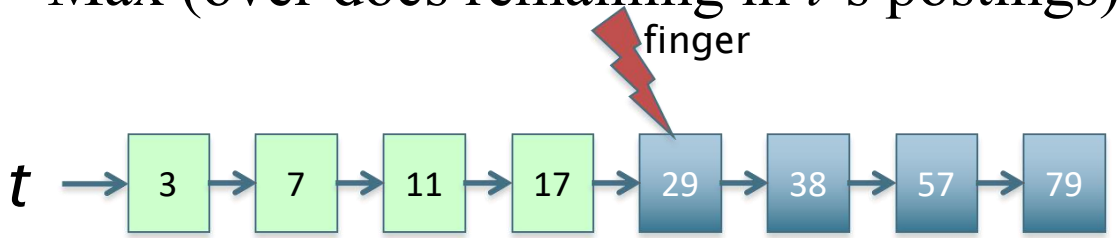
Broder et al. Efficient Query Evaluation using a Two-Level Retrieval Process. CIKM 2003.

WAND索引结构

- 倒排记录表按docID排序
- 假设一个特殊倒排记录遍历器：访问大于等于X的第一个docID
- 典型状态：在每个查询关键字的倒排记录表中，保持一个“指针”（finger）
  - 每个指针仅向（倒排记录表）右移，即指向更高的docID
- 不变-低于任何指针的所有docID已*被处理*，即
  - 这些docID已被去除，或
  - 这些docID的cosine评分已计算完毕

上限（Upper Bound, UB）

- 在任何时候，对于每个查询关键字t，维持一个在指针右边的文档的得分上限UB_t(根据原始词频即可判断)
  - Max (over docs remaining in *t*'s postings) of w_t(doc)

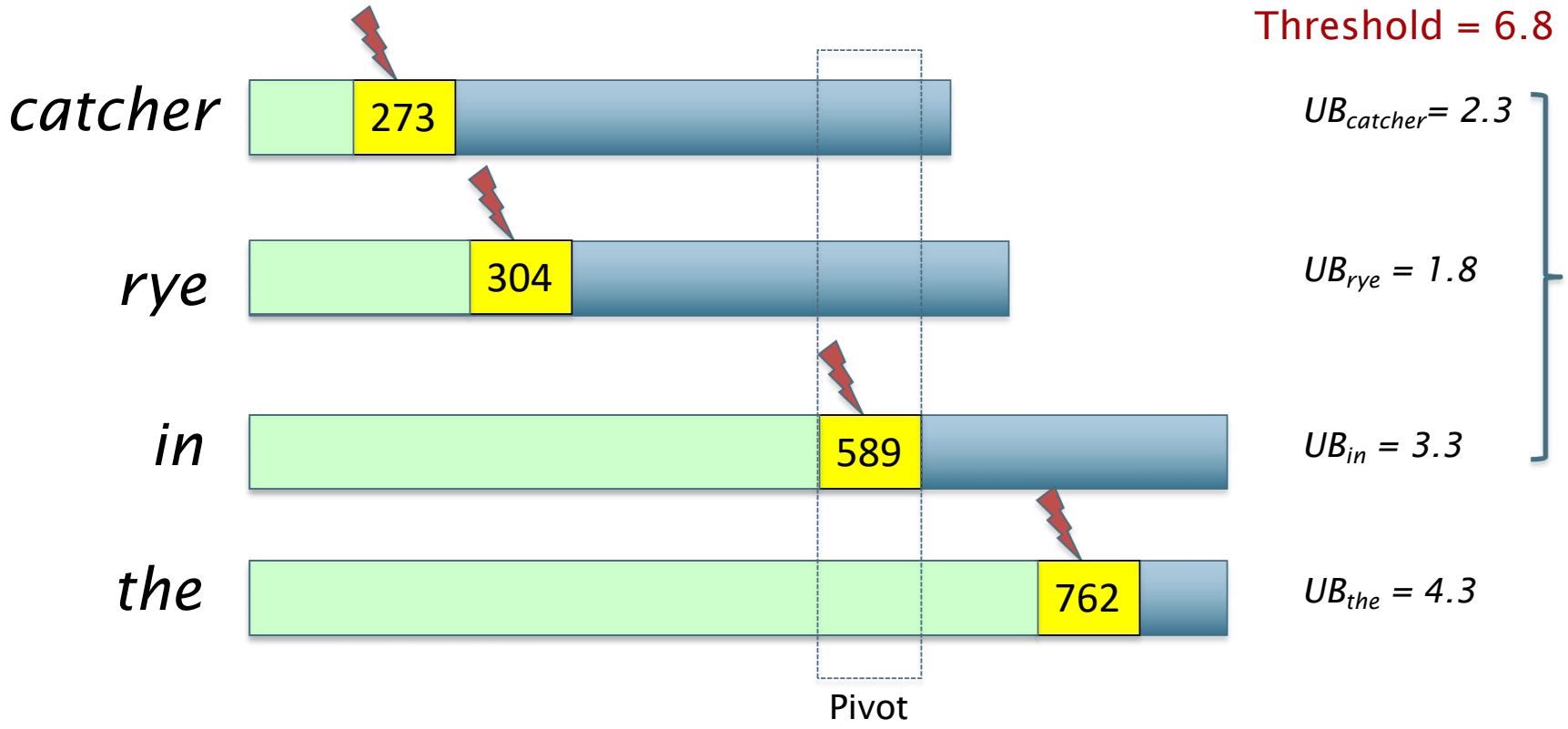


$UB_t = w_t(38)$

随着指针右移，UB下降

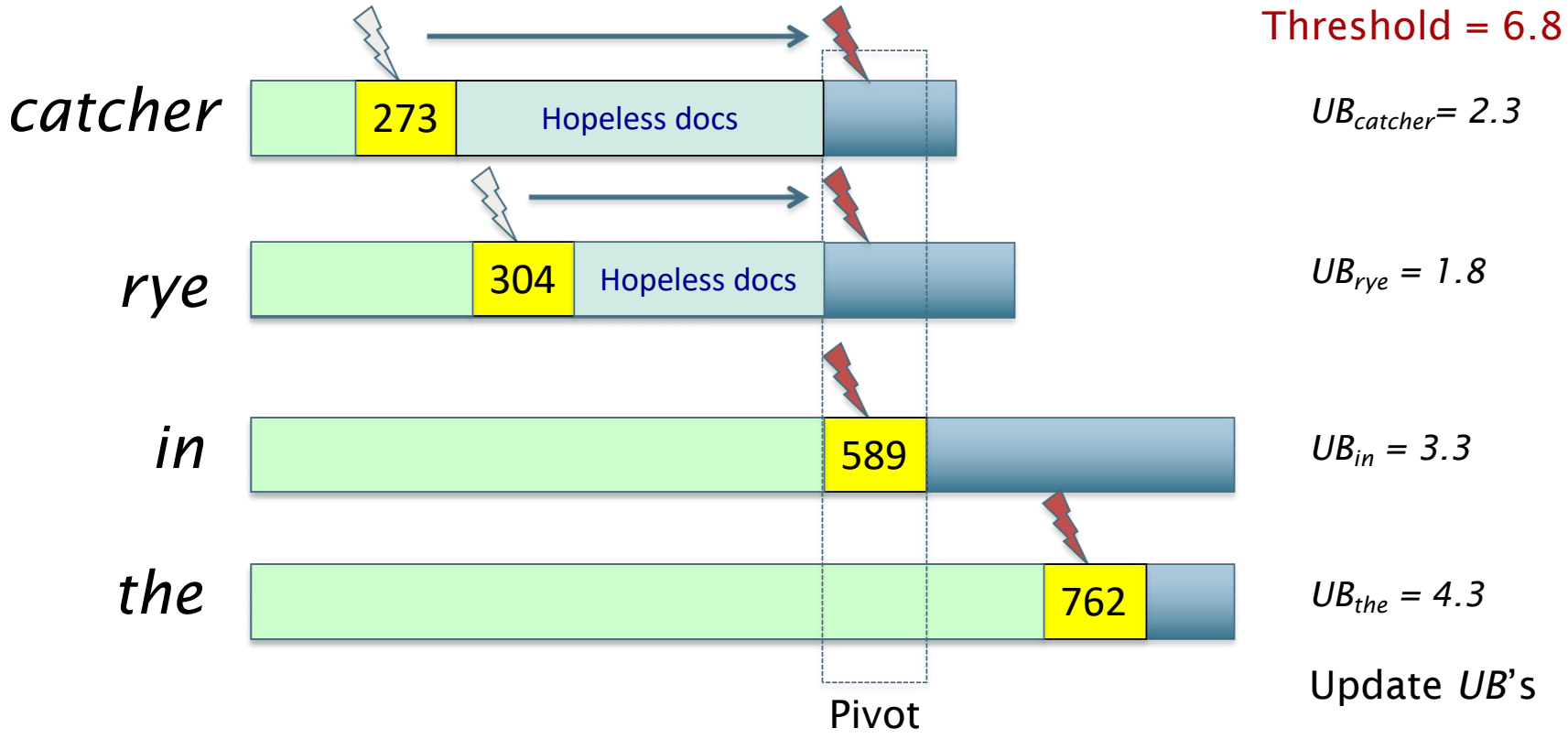
求支点（Pivoting）

- 查询: *catcher in the rye*
- 若当前指针位置如下



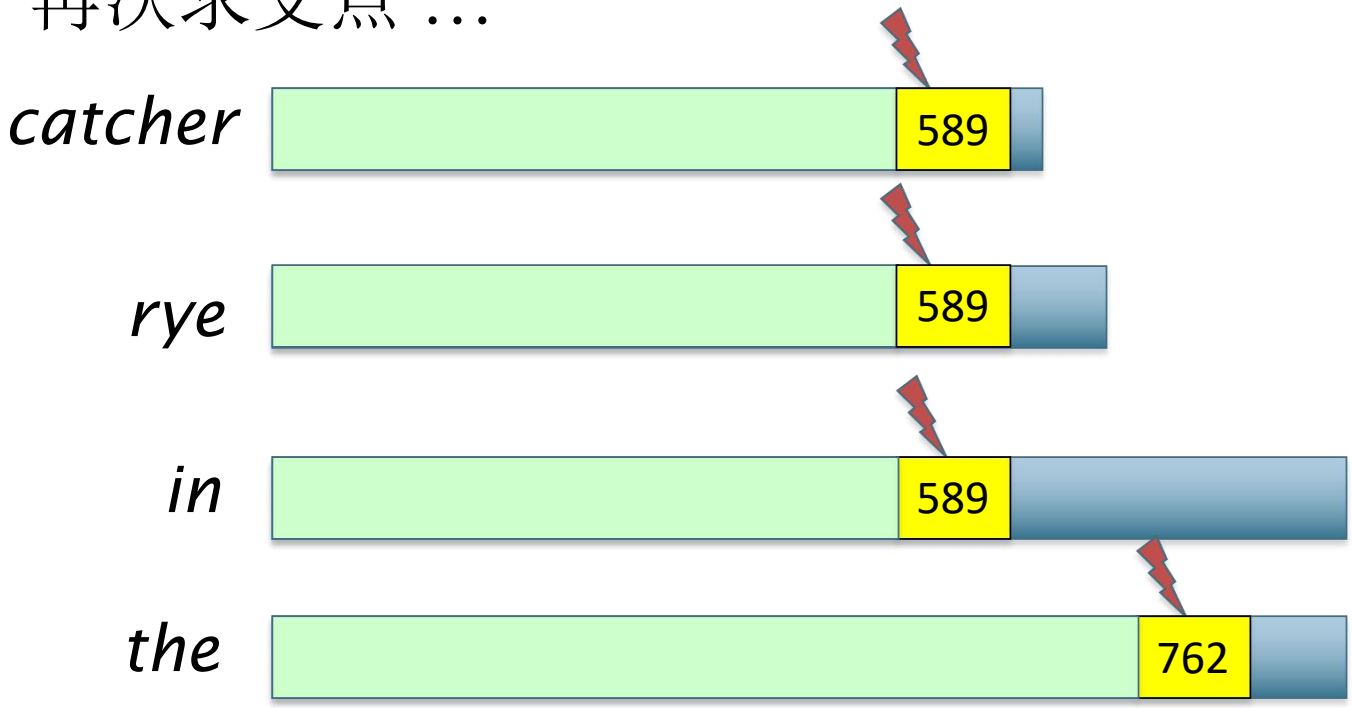
去除评分不可能超过阈值的文档

- 关键字按指针位置排序
- 将589左侧的指针移至支点589 或更靠右的位置



如果需要，计算 589的评分

- 如果文档589在大多数查询关键字的倒排记录表中出现，计算它的完整cosine评分，否则一些指针会处于589右侧
- 再次求支点 …



WAND 总结

- 实验表明, WAND 可以降低 90% 以上的评分计算开支
  - 对长查询的处理的效率提升更显著
- WAND并非仅仅适用于cosine评分排序
  - 适用于任何逐词*附加*的评分方法
- WAND 及其不同的改进版能够满足 安全排序（Safe Ranking, 即精确排序）
  - 同时也可以在此基础上设计更快但是非精确的评分算法

Shuai Ding, Torsten Suel: Faster top-k document retrieval using block-max indexes. SIGIR 2011: 993-1002

提纲

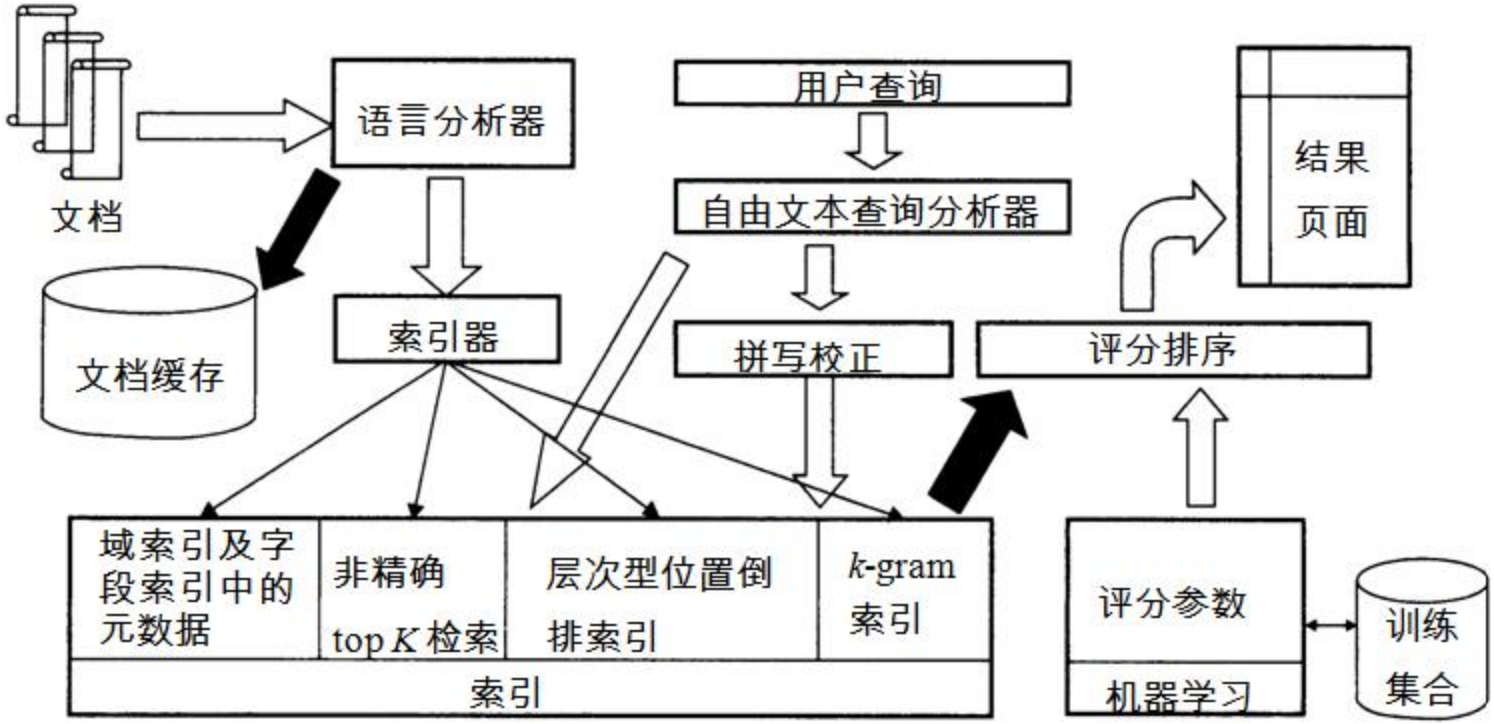
① 上一讲回顾

② 结果排序的动机

③ 结果排序的实现

④ 完整的搜索系统

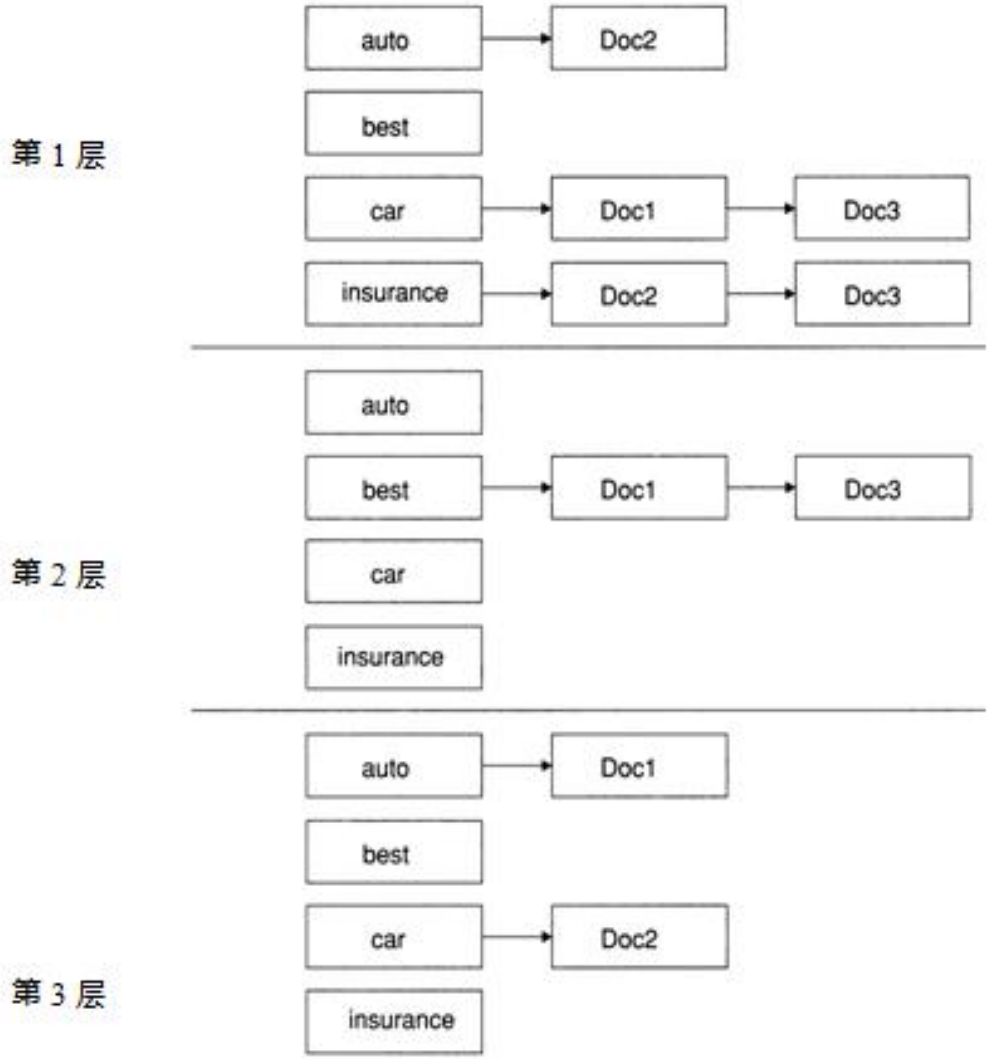
完整的搜索系统示意图


文档
语言分析器
用户查询
自由文本查询分析器
结果
页面
文档缓存
索引器
拼写校正
评分排序
域索引及字
段索引中的
元数据
非精确
top K 检索
层次型位置倒
排索引
k-gram
索引
索引
评分参数
机器学习
训练
集合

多层次索引

- 基本思路:
  - 建立多层索引，每层对应索引词项的重要性
  - 查询处理过程中，从最高层索引开始
  - 如果最高层索引已经返回至少k (比如, $k = 100$)个结果，那么停止处理并将结果返回给用户
  - 如果结果 $< k$ 篇文档，那么从下一层继续处理，直至索引用完或者返回至少k 个结果为止
- 例子：两层的系统
  - 第1层: 所有标题的索引
  - 第2层: 文档剩余部分的索引
  - 标题中包含查询词的页面相对于正文包含查询词的页面而言，排名更应该靠前

多层次索引的例子


第 1 层
auto
Doc2
best
car
Doc1
Doc3
insurance
Doc2
Doc3
第 2 层
auto
best
Doc1
Doc3
car
insurance
第 3 层
auto
Doc1
best
car
Doc2
insurance

多层次索引

- 大家相信，Google (2000/01)搜索质量显著高于其他竞争者的一个主要原因是使用了多层次索引
- (当然还有PageRank、锚文本以及邻近限制条件的使用)

搜索系统组成部分(已介绍)

- 文档预处理 (语言及其他处理)
- 位置信息索引
- 多层次索引
- 拼写校正
- *k*-gram索引(针对通配查询和拼写校正)
- 查询处理
- 文档评分
- 以词项为单位的处理方式

搜索系统组成部分(未介绍)

- 文档缓存(cache): 用它来生成文档摘要(snippet)
- 域索引：按照不同的域进行索引，如文档正文，文档中所有高亮的文本，锚文本、元数据字段中的文本等等
- 基于机器学习的排序函数
- 邻近式排序 (如, 查询词项彼此靠近的文档的得分应该高于查询词项距离较远的文档)
- 查询分析器

向量空间检索与其他方法的融合

- 如何将短语检索和向量空间检索融合在一起？
- 我们不想对每个短语都计算其idf值，为什么？
- 如何将布尔检索和向量空间检索融合在一起？
- 例如: “+”-限制条件和“-”-限制条件
- 后过滤很简单，但是效率低下---没有好的解决办法
- 如何将通配符查询融入到向量空间检索中？同样，没有好的解决方法
- 向量空间模型中的查询和文档向量均为高维稀疏向量，后面会介绍利用无监督学习方法训练低维密集向量的方法

本讲内容

- 排序的重要性：从用户的角度来看(Google的用户研究结果)
- 另一种长度归一化: 回转(Pivoted)长度归一化
- 排序实现
- 完整的搜索系统

参考资料

- 《信息检索导论》第6、7 章
- http://ifnlp.org/ir
  - How Google tweaks its ranking function
  - Interview with Google search guru Udi Manber
  - Yahoo Search BOSS: Opens up the search engine to developers. For example, you can rerank search results.
  - Compare Google and Yahoo ranking for a query
  - How Google uses eye tracking for improving search

课后练习

- 有待补充