信息组织与检索

第7讲：完整搜索系统中的评分计算

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
- 结果排序的动机
- 再论余弦相似度
- 结果排序的实现
- 完整的搜索系统

提纲

- 上一讲回顾
- 结果排序的动机
- 再论余弦相似度
- 结果排序的实现
- 完整的搜索系统

词项频率tf

- t在 d中的对数词频权重定义如下:

$$w_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{tf}_{t,d} & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$$

- 文档-词项的匹配得分
  - 求和
- 几个概念？文档频率df 文档集频率cf
- 为什么要取对数

idf权重

- df_t是出现词项t的文档数目
- df_t是和词项t的信息量成反比的一个值
- 于是可以定义词项t的idf权重:

$$idf_t = \log_{10} \frac{N}{df_t}$$

(其中N是文档集中文档的数目)

- idf_t是反映词项t的信息量的一个指标

tf-idf权重计算

- 词项的tf-idf权重是tf权重和idf权重的乘积

$$w_{t,d} = (1 + \log \mathrm{tf}_{t,d}) \cdot \log \frac{N}{\mathrm{df}_t}$$

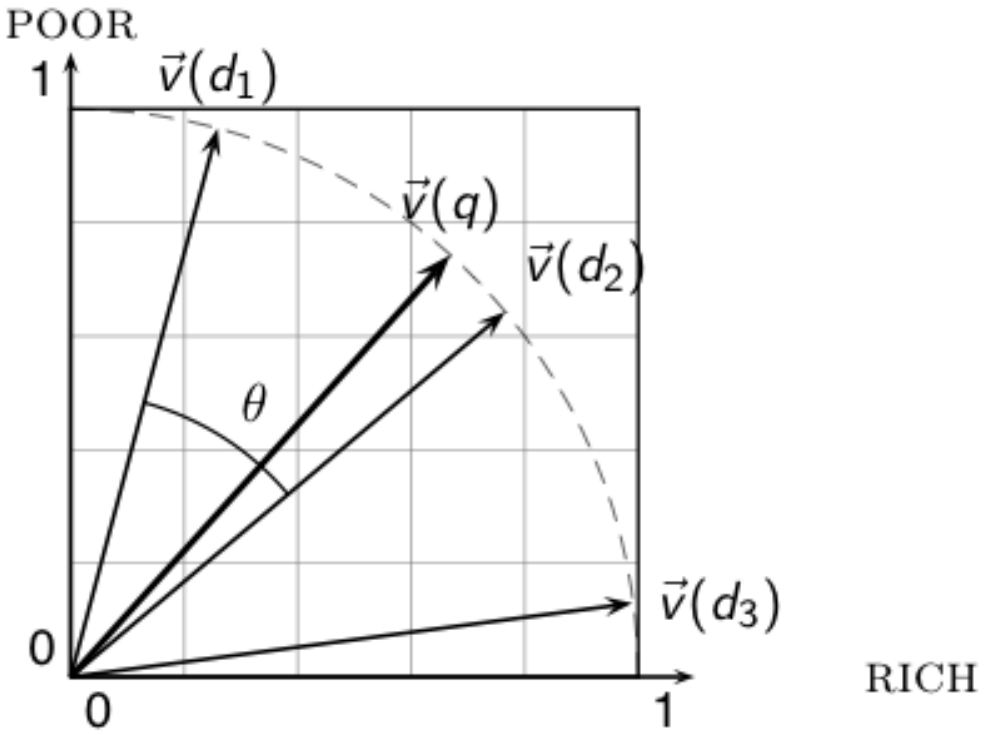
- 信息检索中最出名的权重计算方法之一

查询和文档之间的余弦相似度计算

$$\cos(\vec{q},\vec{d}) = \text{SIM}(\vec{q},\vec{d}) = \frac{\vec{q}\cdot\vec{d}}{|\vec{q}||\vec{d}|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2}\sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

- q_i 是第 i 个词项在查询 q 中的tf-idf权重
- d_i 是第 i 个词项在文档 d 中的tf-idf权重

余弦相似度计算的图示



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

提纲

- 上一讲回顾
- 结果排序的动机
- 再论余弦相似度
- 结果排序的实现
- 完整的搜索系统

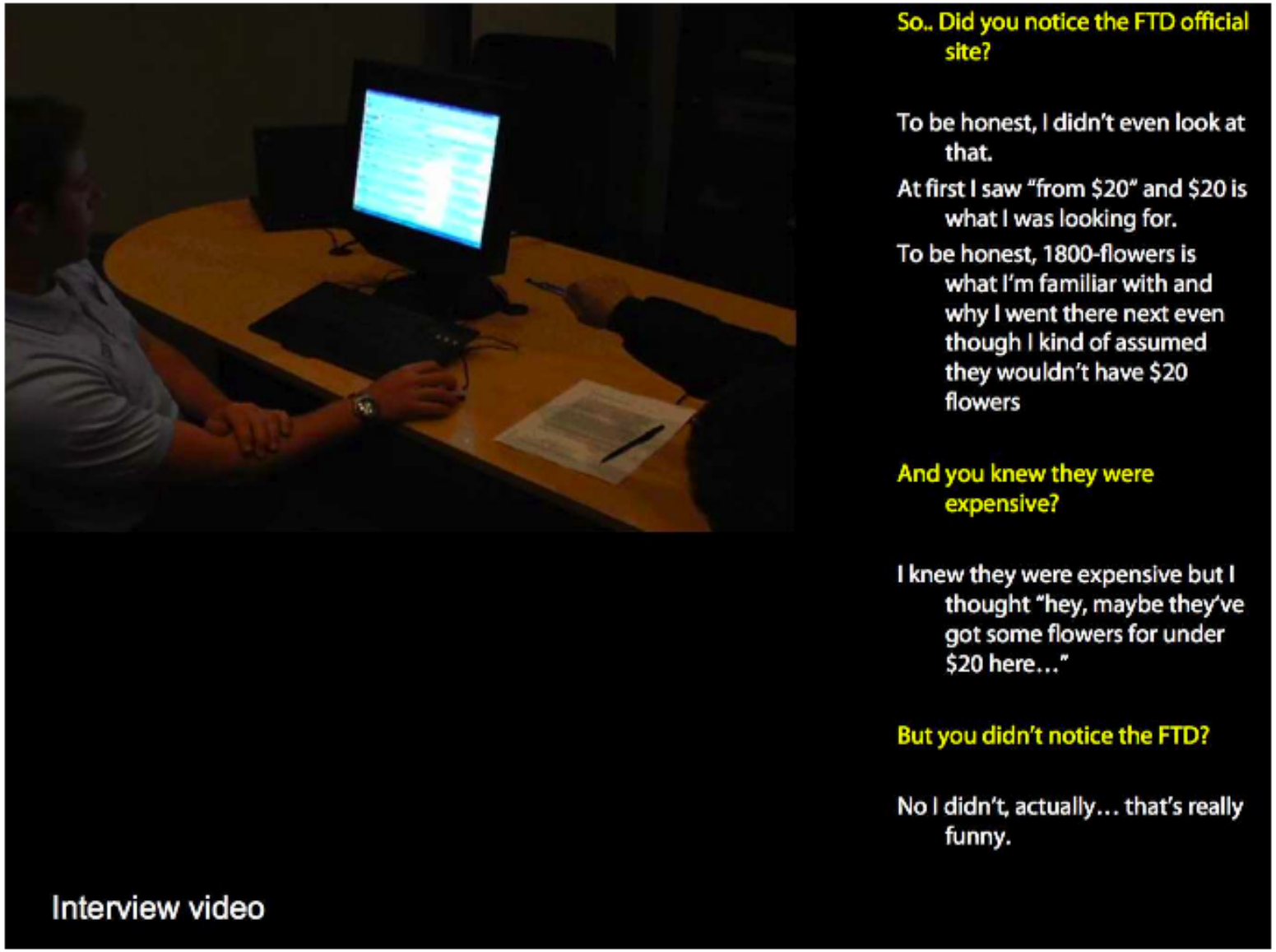
排序的重要性

- 上一讲: 结果不排序，后果很严重
  - 用户只希望看到一些而不是成千上万的结果
  - 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

检索效果的经验性观察方法

- 如何度量排序的重要性?
- 可以在某种受控配置观察下搜索用户的行为
  - 对用户行为进行录像
  - 让他们放声思考Ask them to “think aloud”
  - 访谈
  - 眼球跟踪
  - 计时
  - 记录并对他们的点击计数

用户访谈


So.. Did you notice the FTD official site?
To be honest, I didn't even look at that.
At first I saw "from $20" and $20 is what I was looking for.
To be honest, 1800-flowers is what I'm familiar with and why I went there next even though I kind of assumed they wouldn't have $20 flowers
And you knew they were expensive?
I knew they were expensive but I thought "hey, maybe they've got some flowers for under $20 here…"
But you didn't notice the FTD?
No I didn't, actually… that's really funny.
Interview video

用户对结果的浏览模式

Rapidly scanning the results

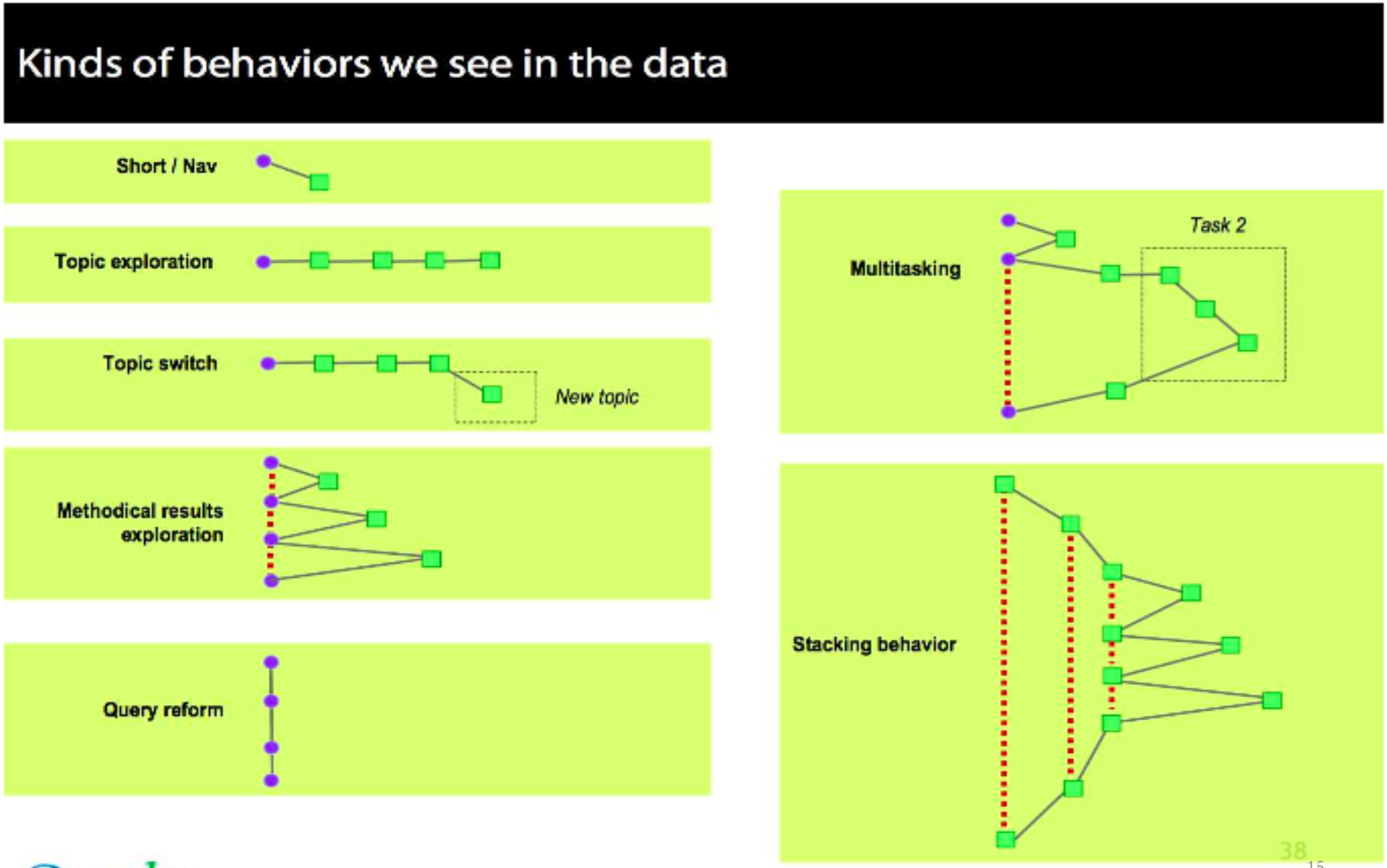
Note scan pattern:

Page 3:
- Result 1
- Result 2
- Result 3
- Result 4
- Result 3
- Result 2
- Result 4
- Result 5
- Result 6 <click>

Q: Why do this?

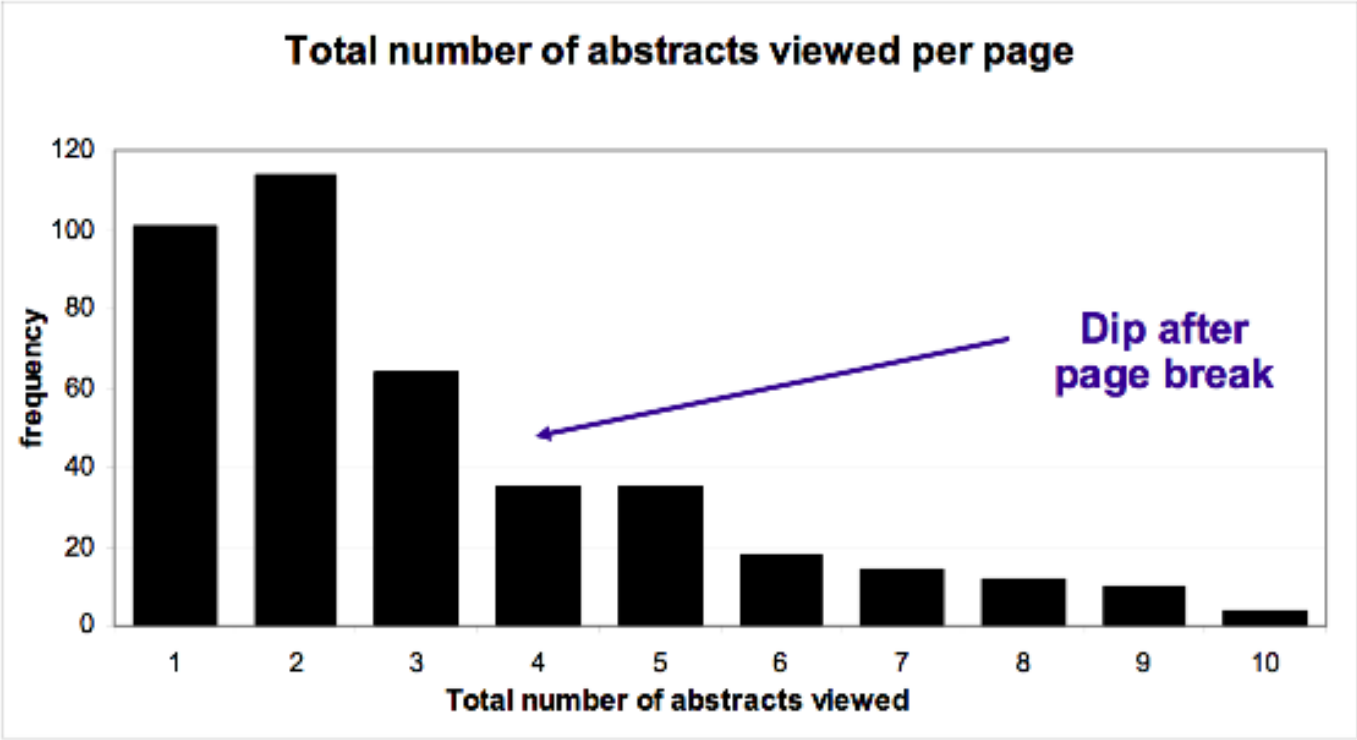
A: What's learned later influences judgment of earlier content.

检索中的用户行为模式


Kinds of behaviors we see in the data
Short / Nav
Topic exploration
Topic switch
New topic
Methodical results exploration
Query reform
Multitasking
Task 2
Stacking behavior

用户浏览的链接数

How many links do users view?

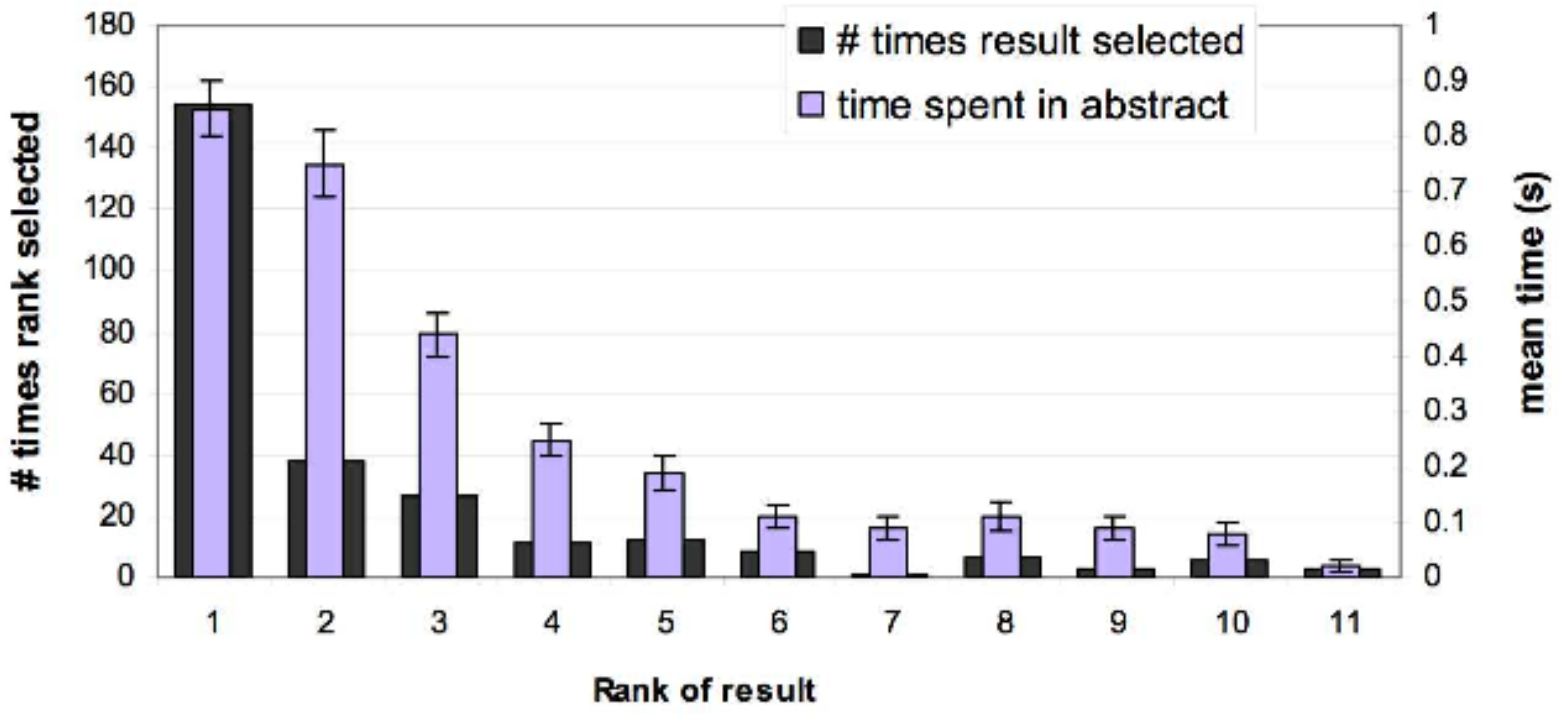


Mean: 3.07 Median/Mode: 2.00

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

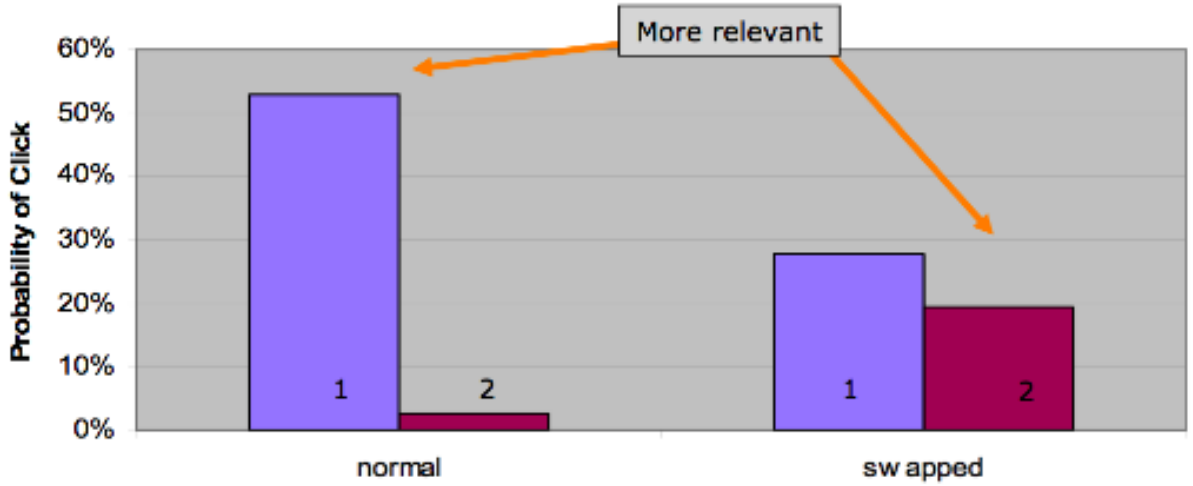
用户通过阅读摘要，来决定是否点击

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



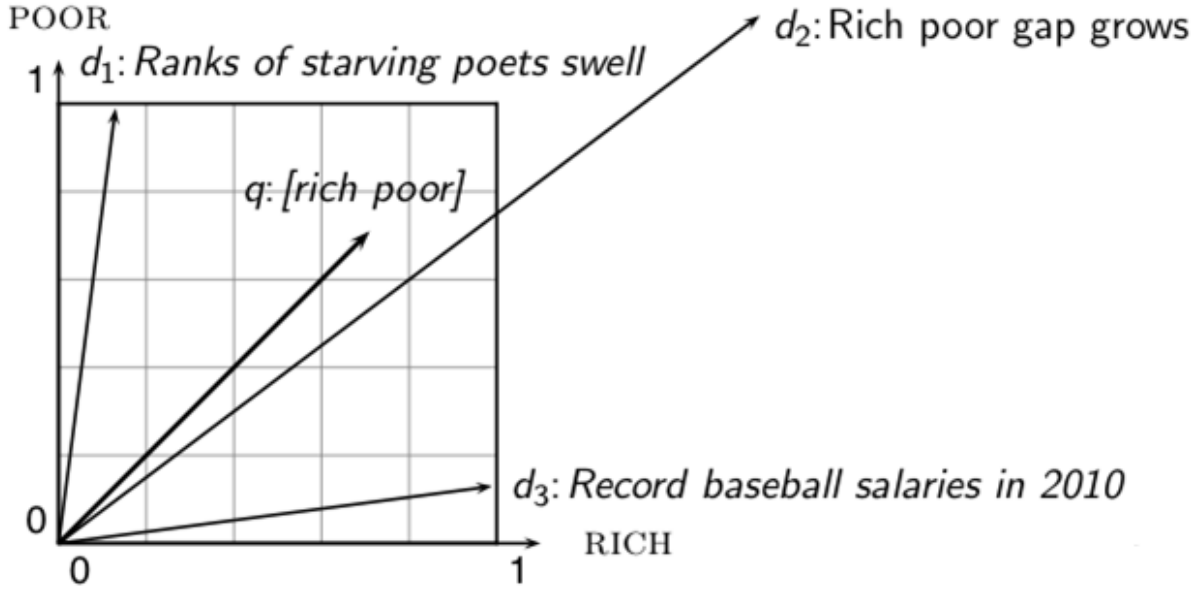
排序的重要性: 小结

- 摘要阅读(Viewing abstracts): 用户更可能阅读前几页(1, 2, 3, 4)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相关，仍然有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

- 上一讲回顾
- 结果排序的动机
- 再论余弦相似度
- 结果排序的实现
- 完整的搜索系统

距离函数不适合度量相似度



- 尽管查询q和文档d_2的内容很相似，但是向量 $\vec{q}$ 和 $\vec{d}_2$ 的欧氏距离却很大。这也是为什么要进行长度归一化的原因，或者说，我们前面采用余弦相似度的原因。

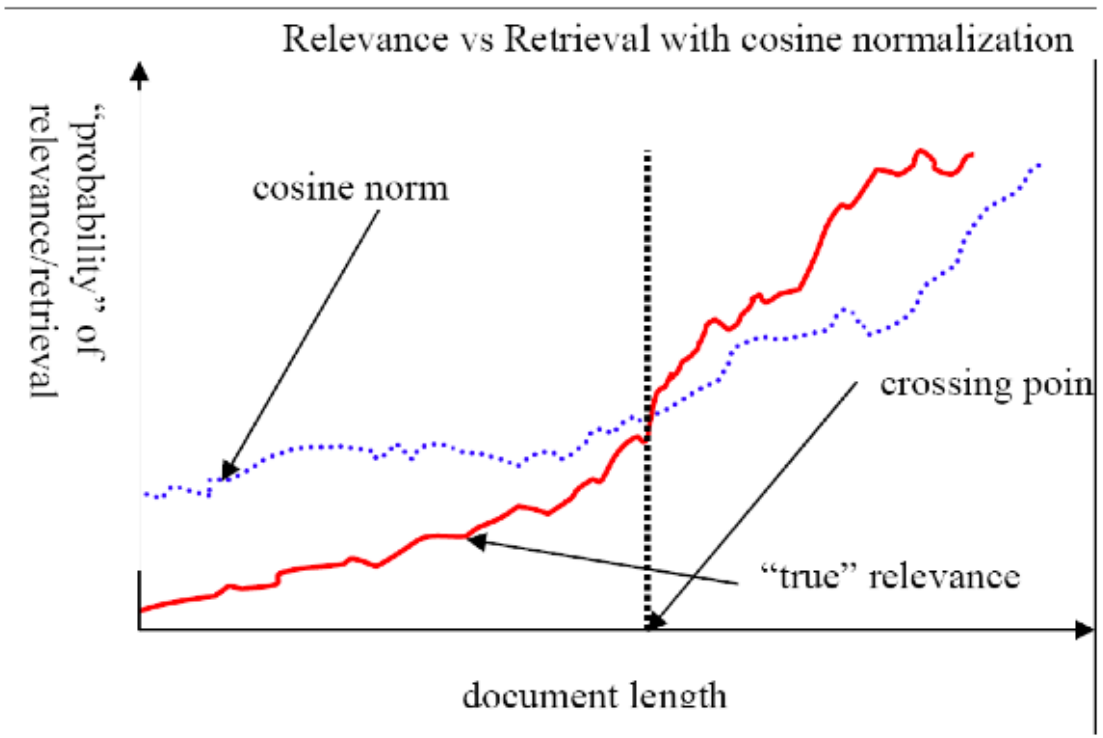
余弦相似度的一个问题

- 查询 q: “anti-doping rules Beijing 2008 olympics”　　反兴奋剂
- 计算并比较如下的三篇文档
  - d1: 一篇有关”anti-doping rules at 2008 Olympics”的中等长度文档
  - d2: 一篇包含d1 以及其他5篇新闻报道的长文档，其中这5篇新闻报道的主题都与Olympics/anti-doping无关
  - d3: 一篇有关”anti-doping rules at the 2004 Athens Olympics“的短文档
- 我们期望的结果是什么？
- 如何实现上述结果？

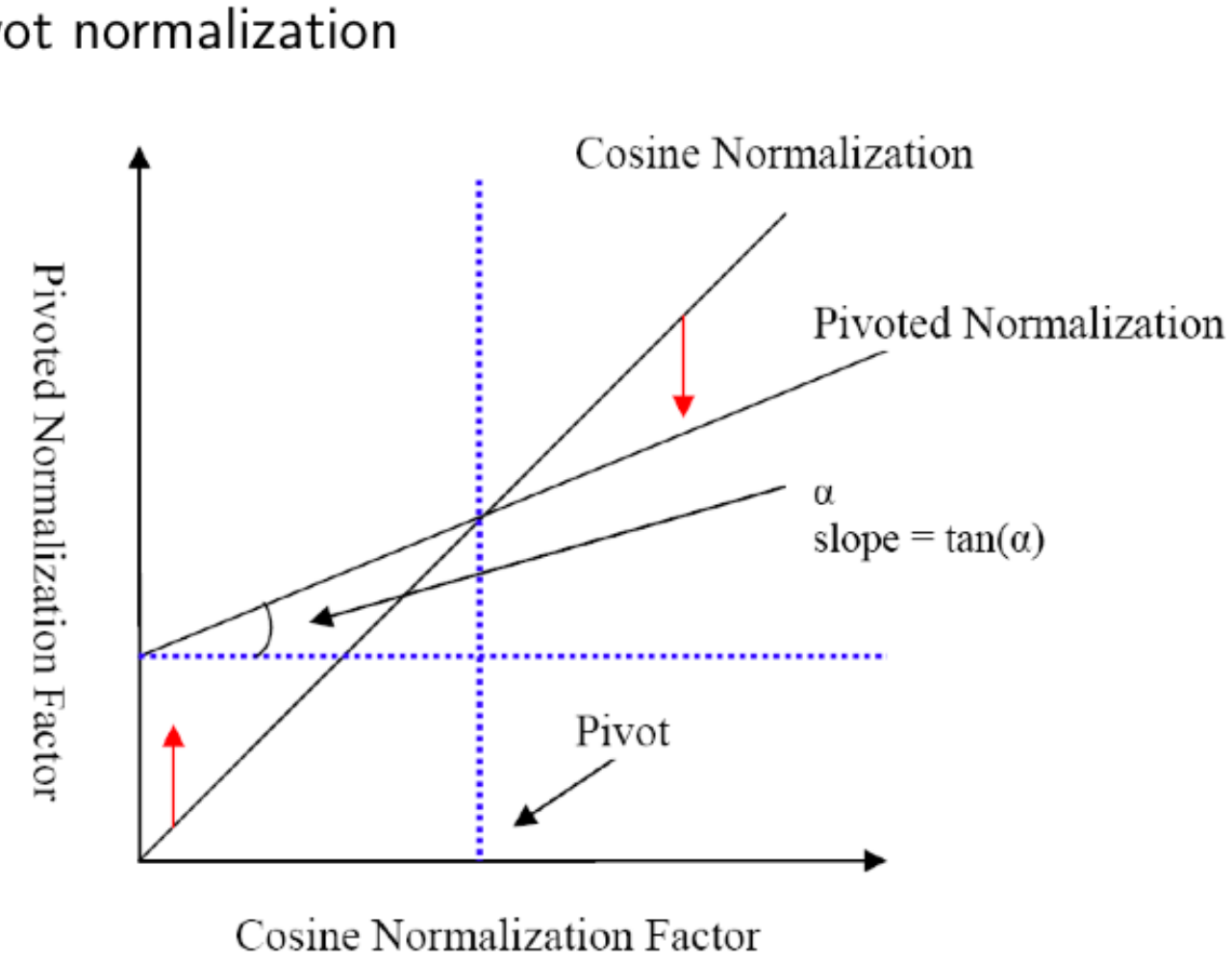
回转归一化

- 余弦归一化倾向于短文档，即对短文档产生的归一化因子太大，而平均而言对长文档产生的归一化因子太小
- 于是可以先找到一个支点(pivot，平衡点)，然后通过这个支点对余弦归一化操作进行线性调整。
- 效果：短文档的相似度降低，而长文档的相似度增大
- 这可以去除原来余弦归一化偏向短文档的问题

预测相关性概率 vs. 真实相关性概率



回转归一化(Pivot normalization)



回转归一化: Amit Singhal的实验结果

Cosine	Pivoted Cosine Normalization				
	Slope				
	0.60	0.65	0.70	**0.75**	0.80
6,526	6,342	6,458	6,574	**6,629**	6,671
0.2840	0.3024	0.3097	0.3144	**0.3171**	0.3162
Improvement	+ 6.5%	+ 9.0%	+10.7%	**+11.7%**	+11.3%

- 结果第一行: 返回的相关文档数目
- 结果第二行: 平均正确率
- 结果第三行: 平均正确率的提高百分比

相关论文

Pivoted document length normalization

A Singhal, C Buckley, M Mitra - ACM SIGIR Forum, 2017 - dl.acm.org

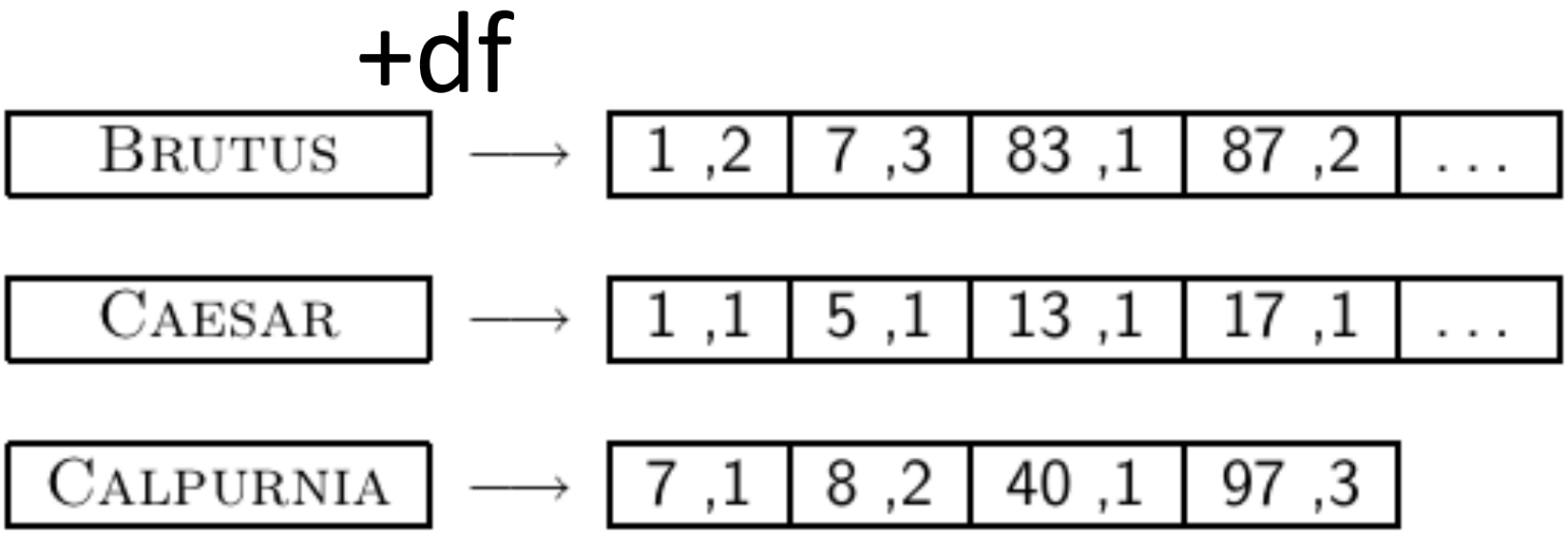
Automatic information retrieval systems have to deal with documents of varying lengths in a text collection. Document length normalization is used to fairly retrieve documents of all lengths. In this study, we ohserve that a normalization scheme that retrieves documents of all ...

☆ 99 Cited by 1261 Related articles All 19 versions »

提纲

- 上一讲回顾
- 结果排序的动机
- 再论余弦相似度
- 结果排序的实现
- 完整的搜索系统

词项频率tf也存入倒排索引中



倒排索引中的词项频率存储

- 每条倒排记录中，除了docID 还要存储tf，df
- 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
  - 为什么?
  - 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

余弦相似度计算算法

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[]
```

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.

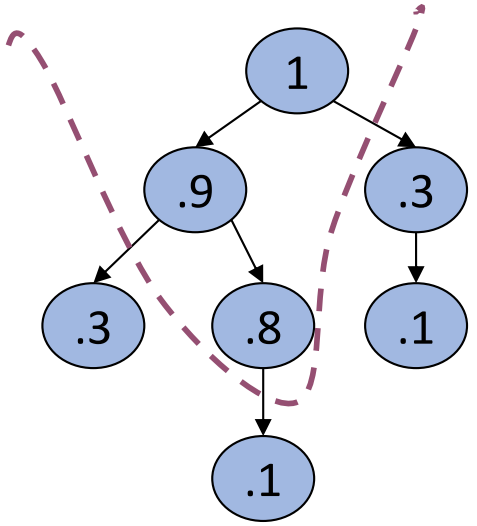
精确top k检索加速方法二：堆法N中选K

- 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

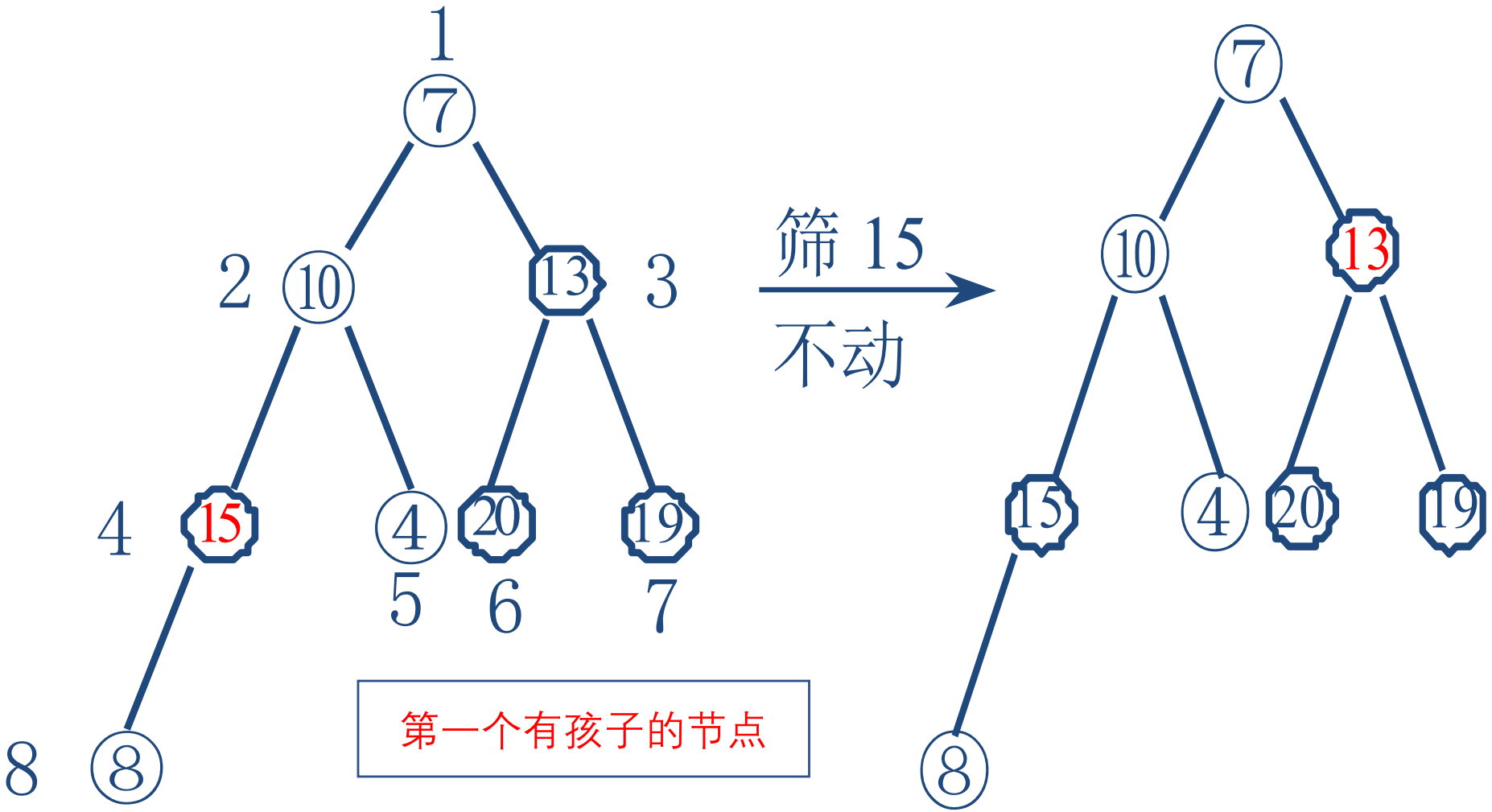
堆方法

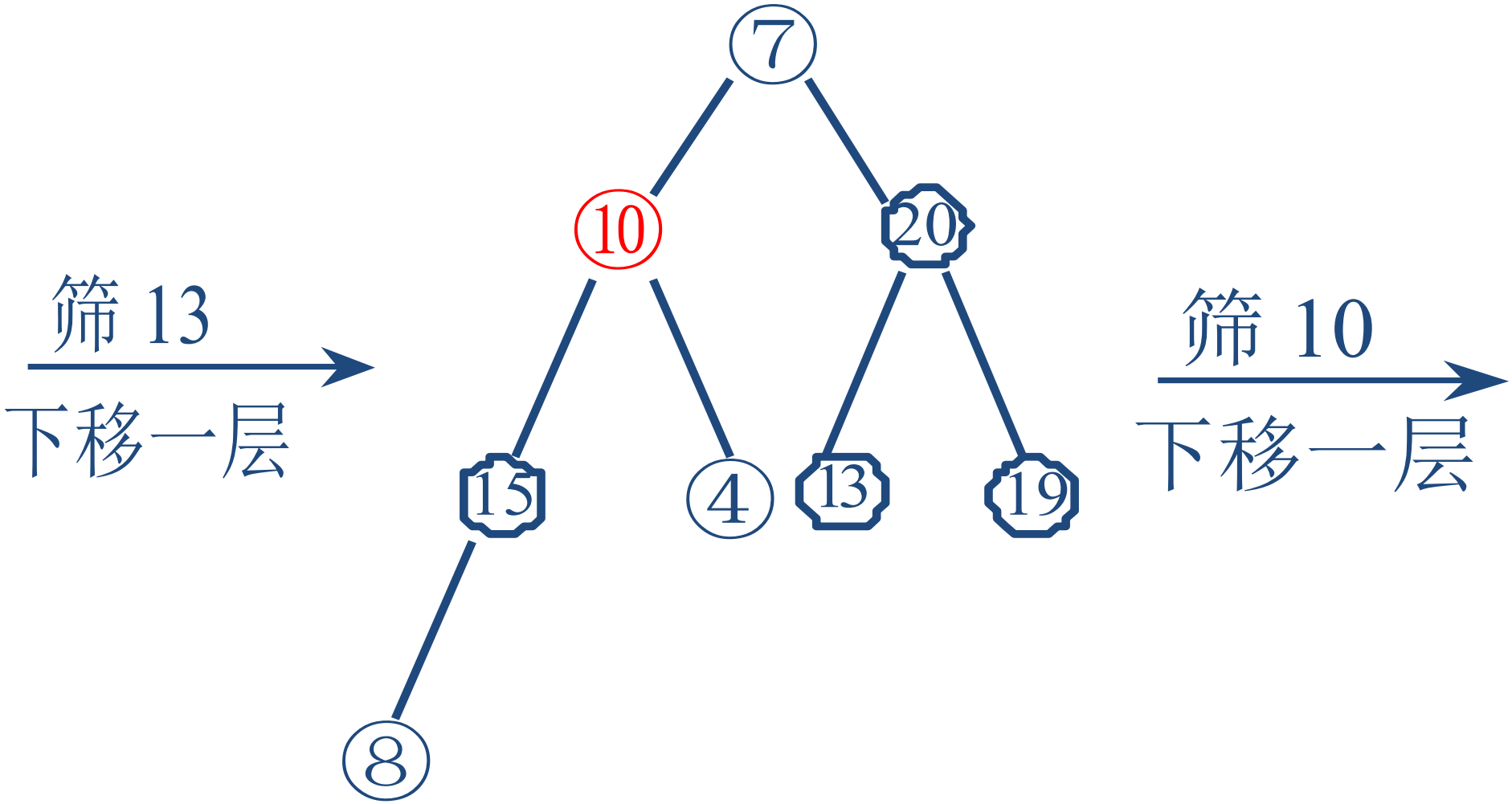
- 堆：二叉树的一种，每个节点上的值> 子节点上的值 (Max Heap)
- 堆构建：需要 2J 次操作
- 选出前K个结果：每个结果需要2log J 步
- 如果 J=1M, K=100, 那么代价大概是全部排序代价的10%

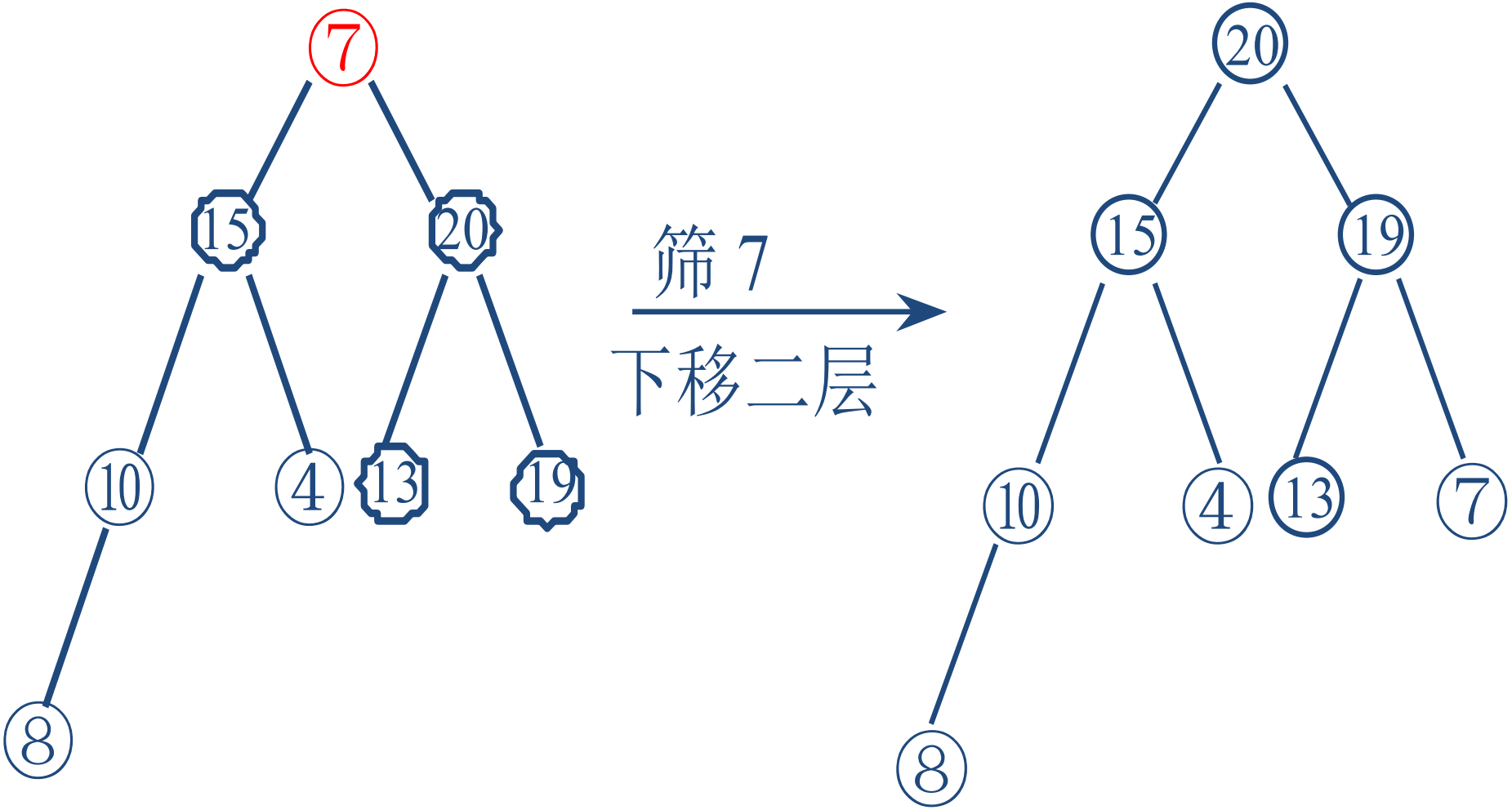


(最大)堆构建样例(筛选shift法-摘自网上课件)

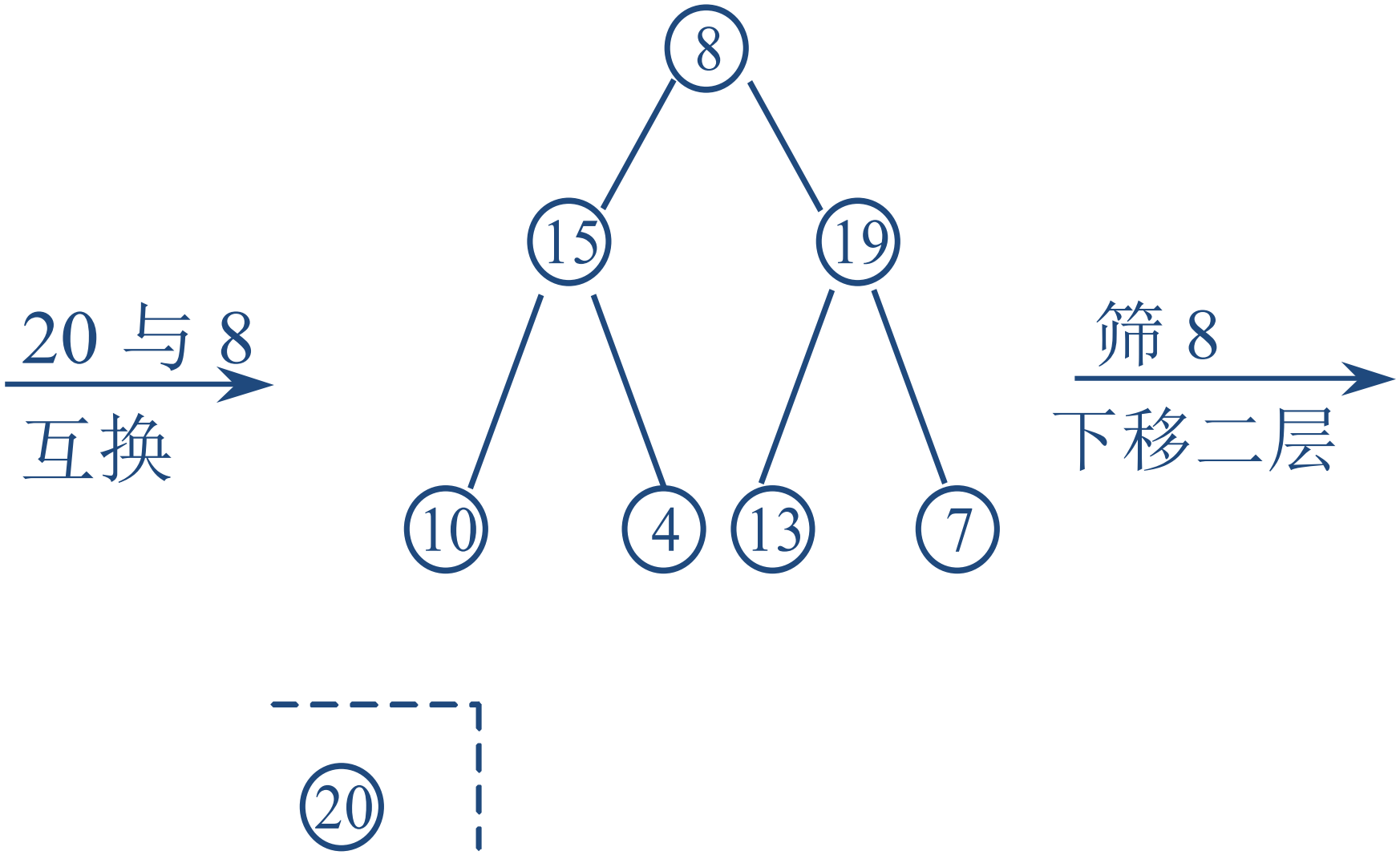
- 7，10，13，15，4，20，19，8（数据个数n=8）。

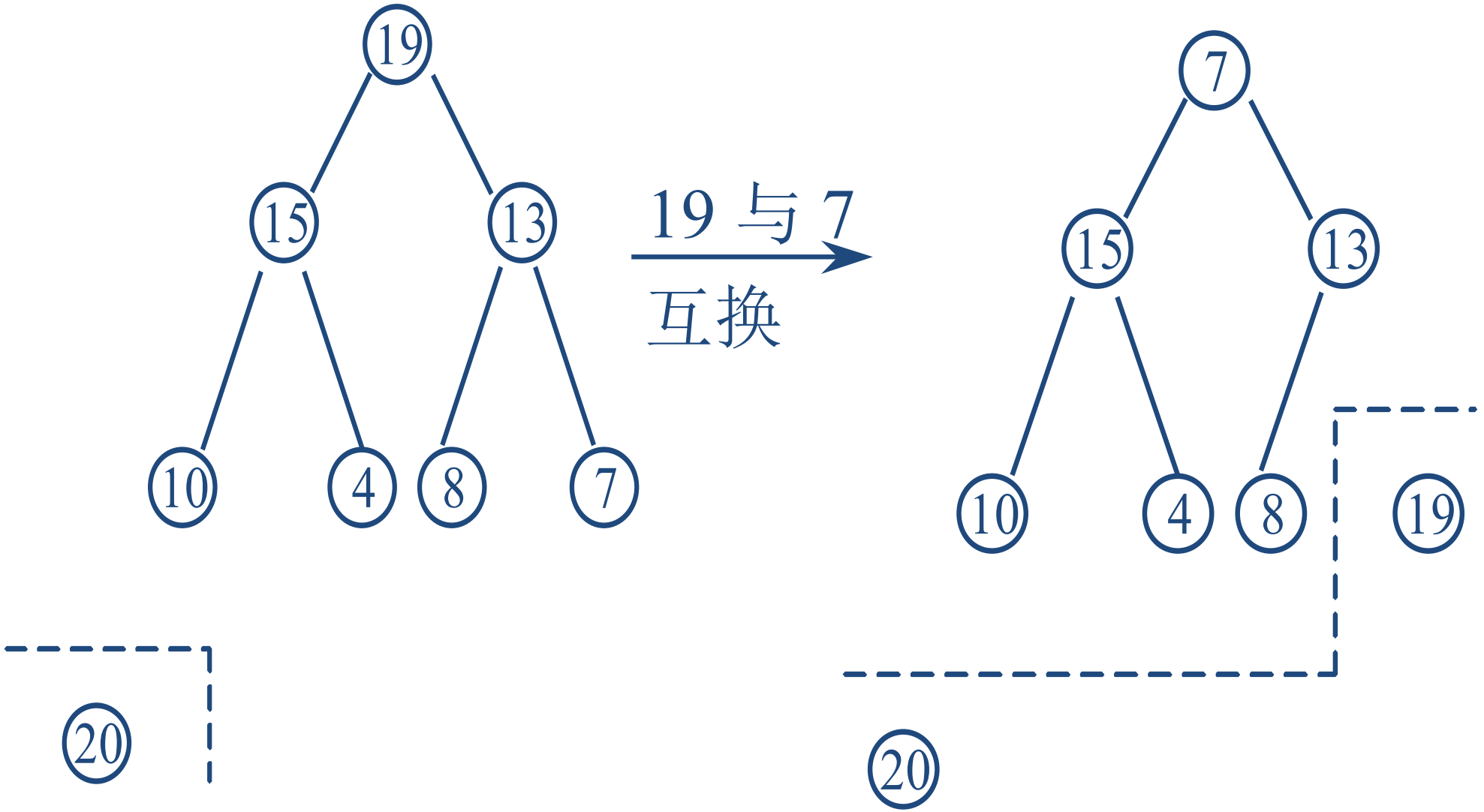


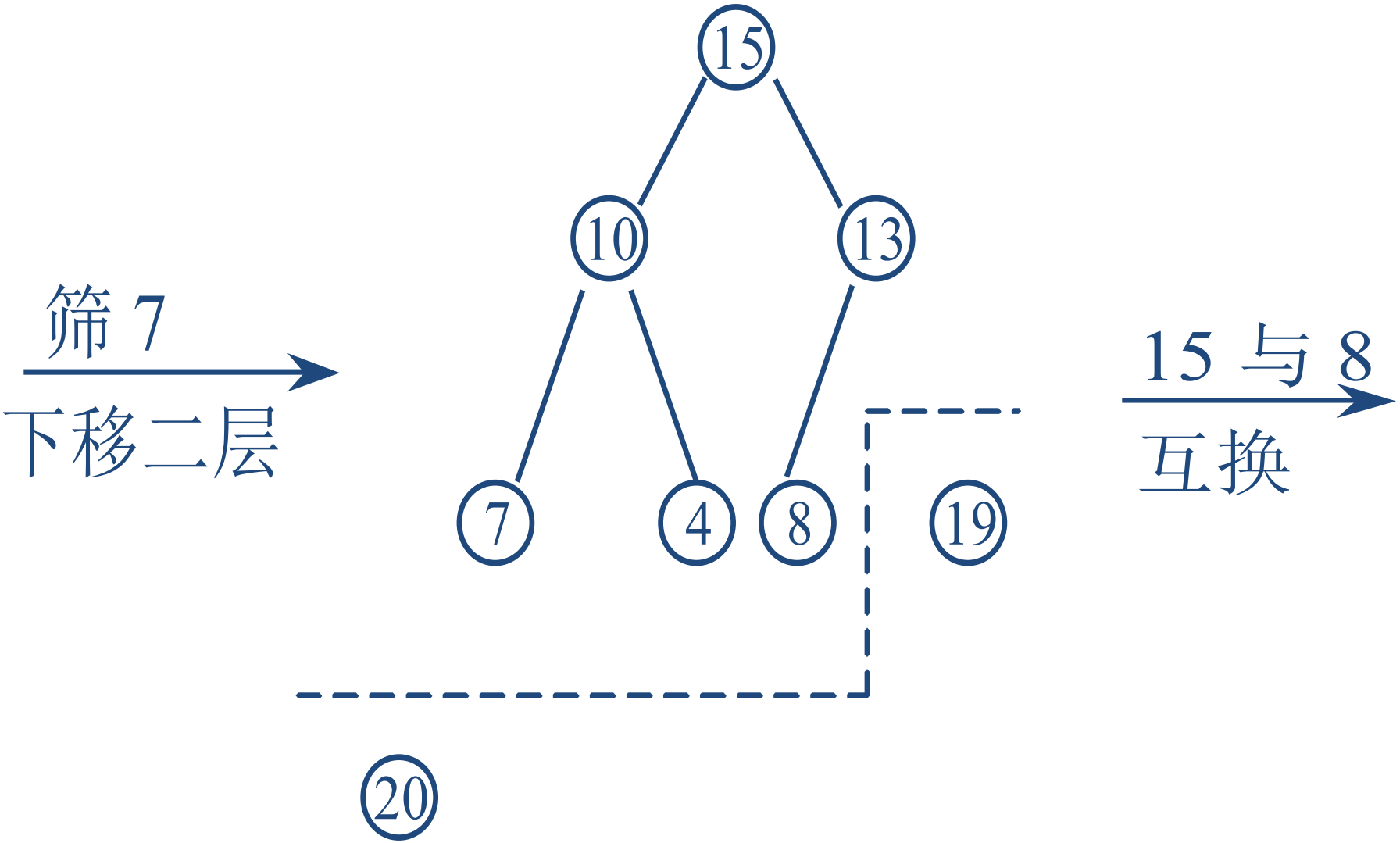

筛 13
下移一层
7
10
20
15
4
13
19
8
筛 10
下移一层

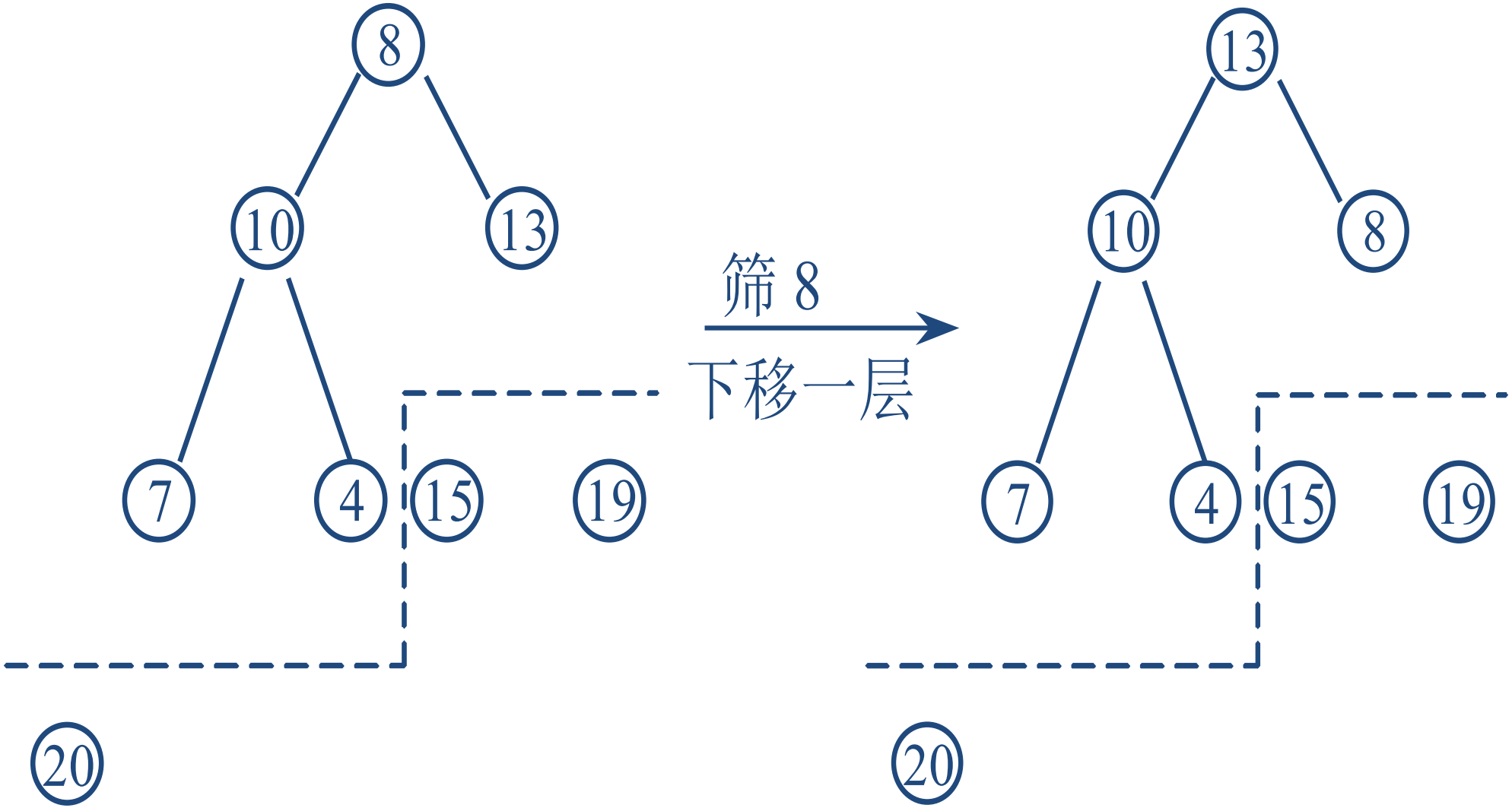

7
15
20
10
4
13
19
8
筛 7
下移二层
20
15
19
10
4
13
7
8

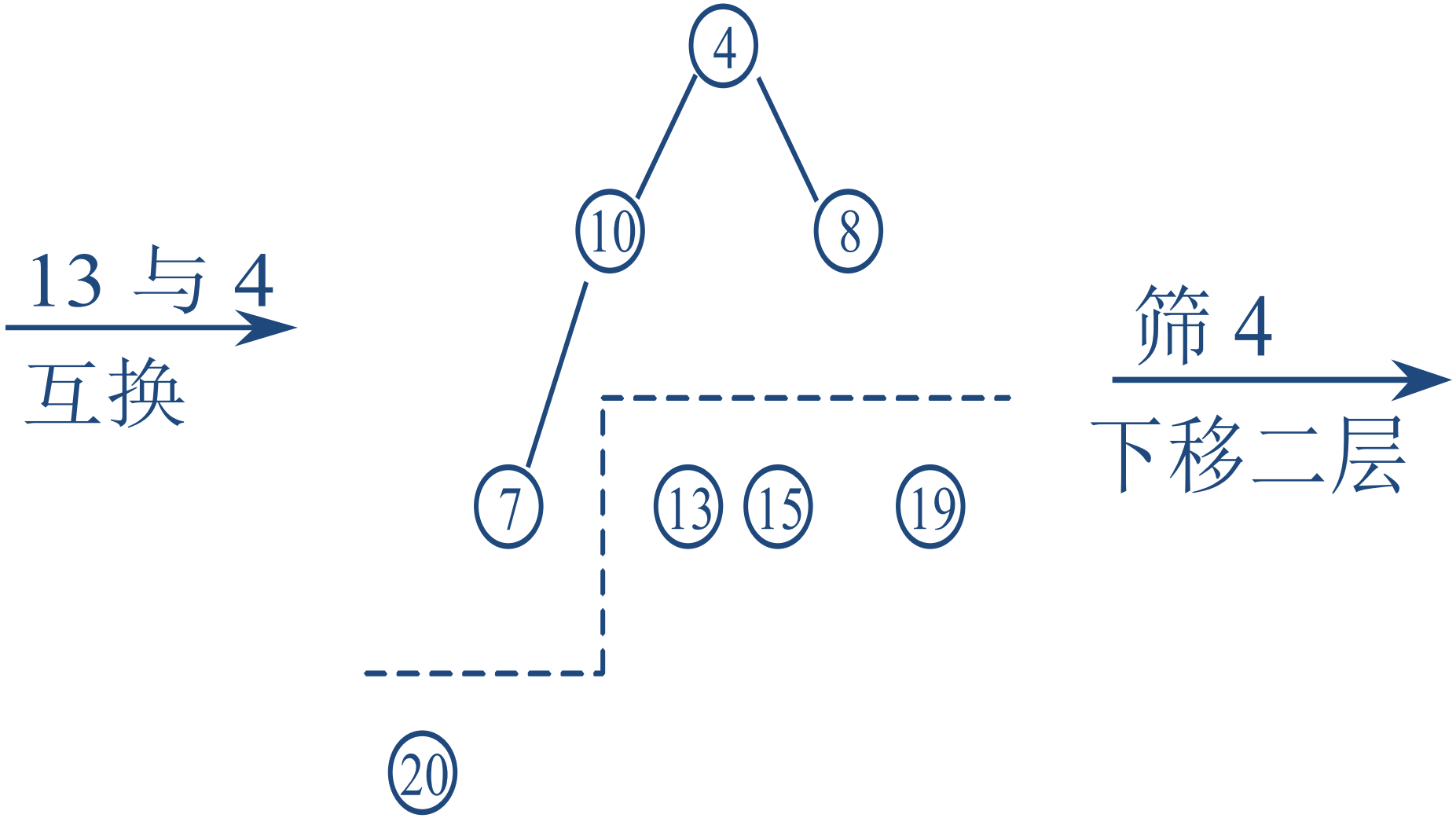
利用堆选出Top K (=4)




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


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

(最大)堆构建样例（课堂练习）

- 7，10，13，15，4，20，19，8（数据个数n=8）。

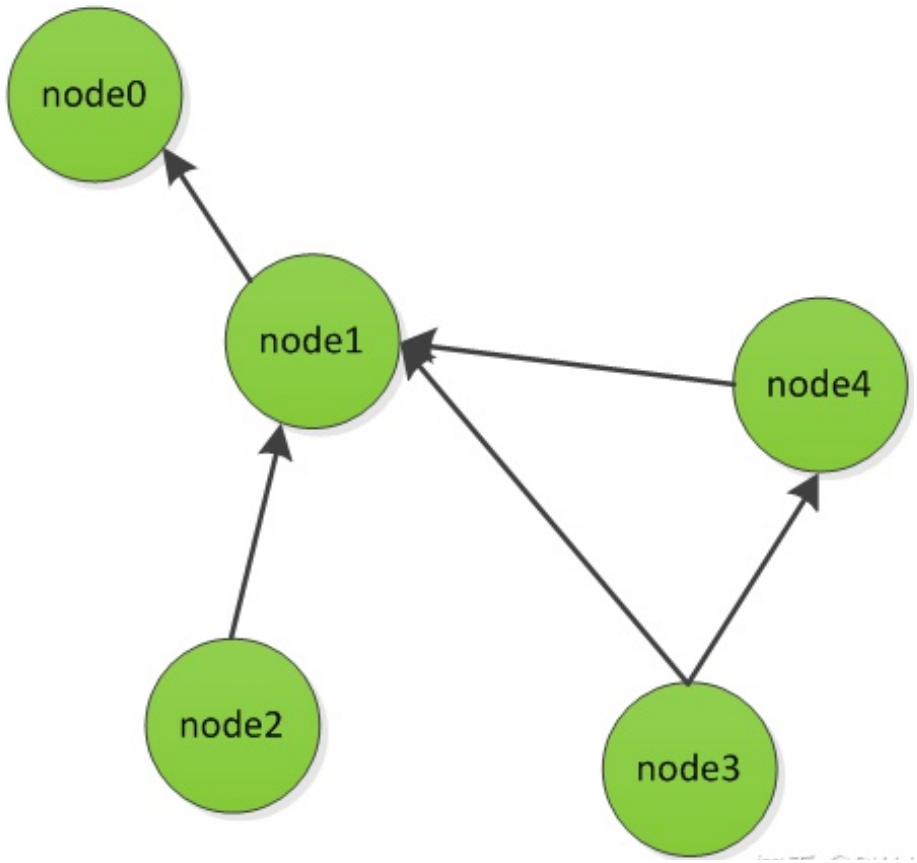
构建堆
取最大值
重构堆

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（下一页）
  - 于是可以将文档按照PageRank排序

  g(d1) > g(d2) > g(d3) > . . .
- 将PageRank和余弦相似度线性组合得到文档的最后得分

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



- 应用场景
  - 网页
  - 论文
  - 微博
- 重要性排序
  - 传播力最强
- 思想
  - 通过看一个人的朋友来分析这个人

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(以文档为单位的计算，document-at-a-time)。
    如果当前处理的文档的 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
- 一个自然而言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)

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，为什么不是tfidf值最高的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

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)

- 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

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-idftd 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，分别成为高端表和低端表
  - 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
  - $\mathrm{wf}_{t,d}$ 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)

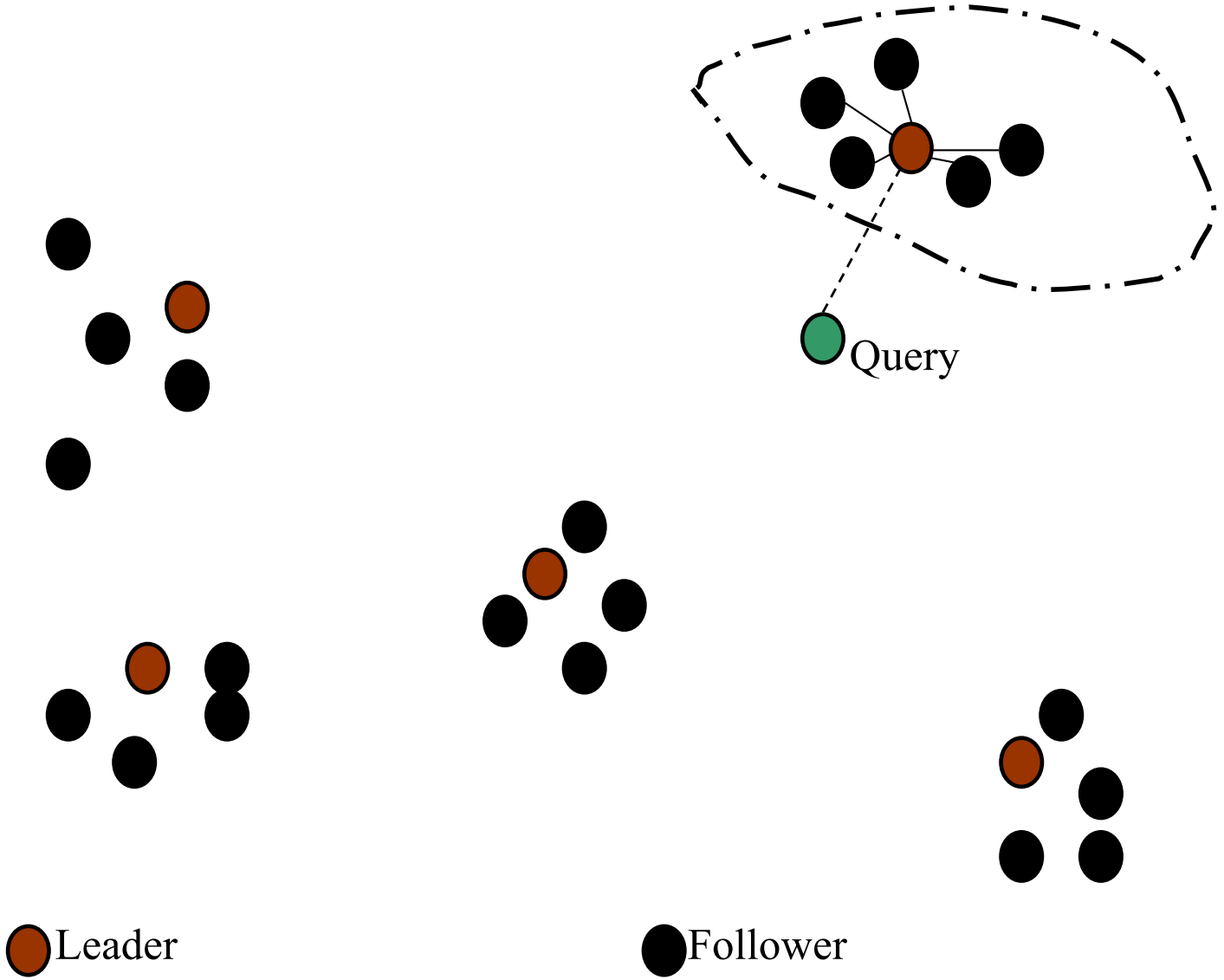
- 随机选 √N 篇文档作为先导者

- 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



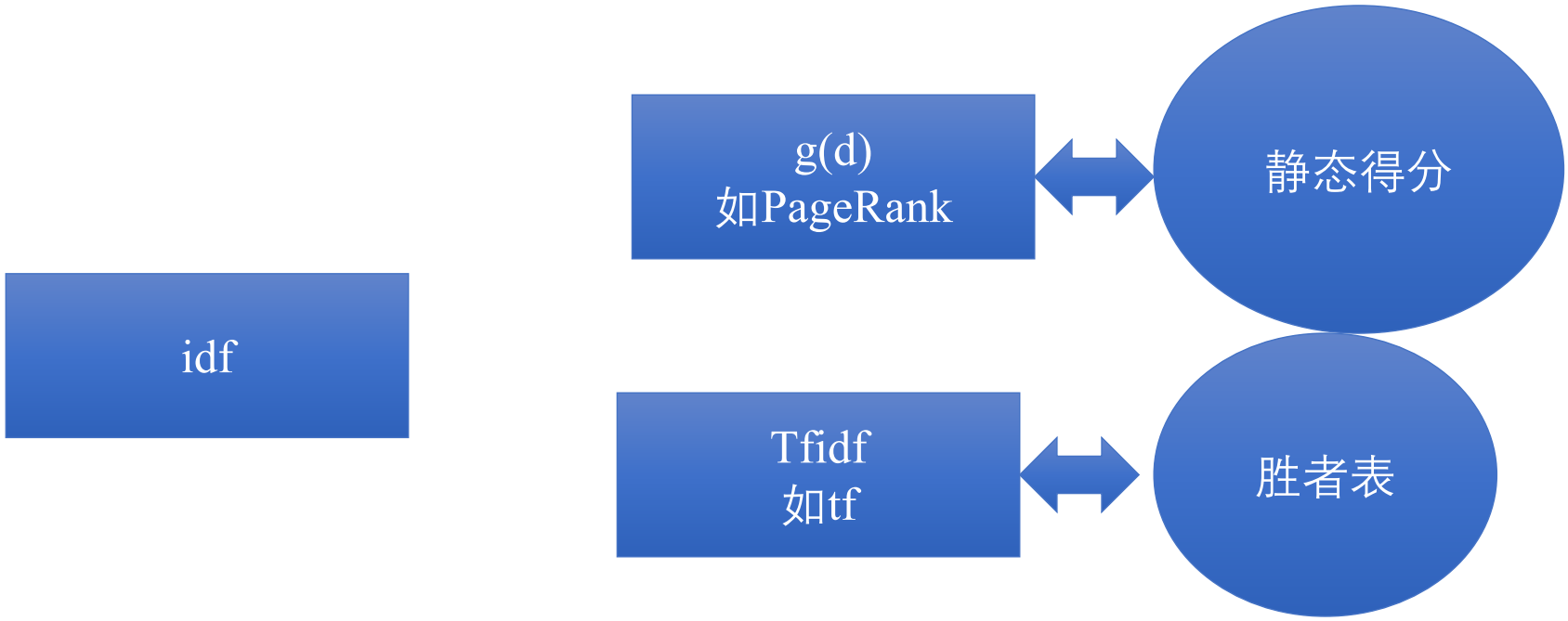
为什么采用随机抽样?

- 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

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

- 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

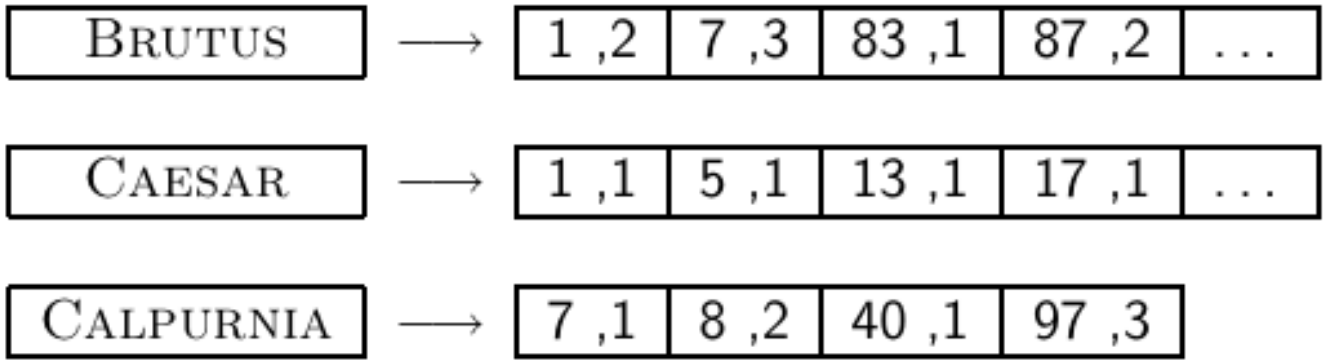
```
CosineScore(q)
 1  float Scores[N] = 0
 2  float Length[N]
 3  for each query term t
 4  do calculate w_{t,q} and fetch postings list for t
 5      for each pair(d, tf_{t,d}) in postings list
 6      do Scores[d]+ = w_{t,d} × w_{t,q}
 7  Read the array Length
 8  for each d
 9  do Scores[d] = Scores[d]/Length[d]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

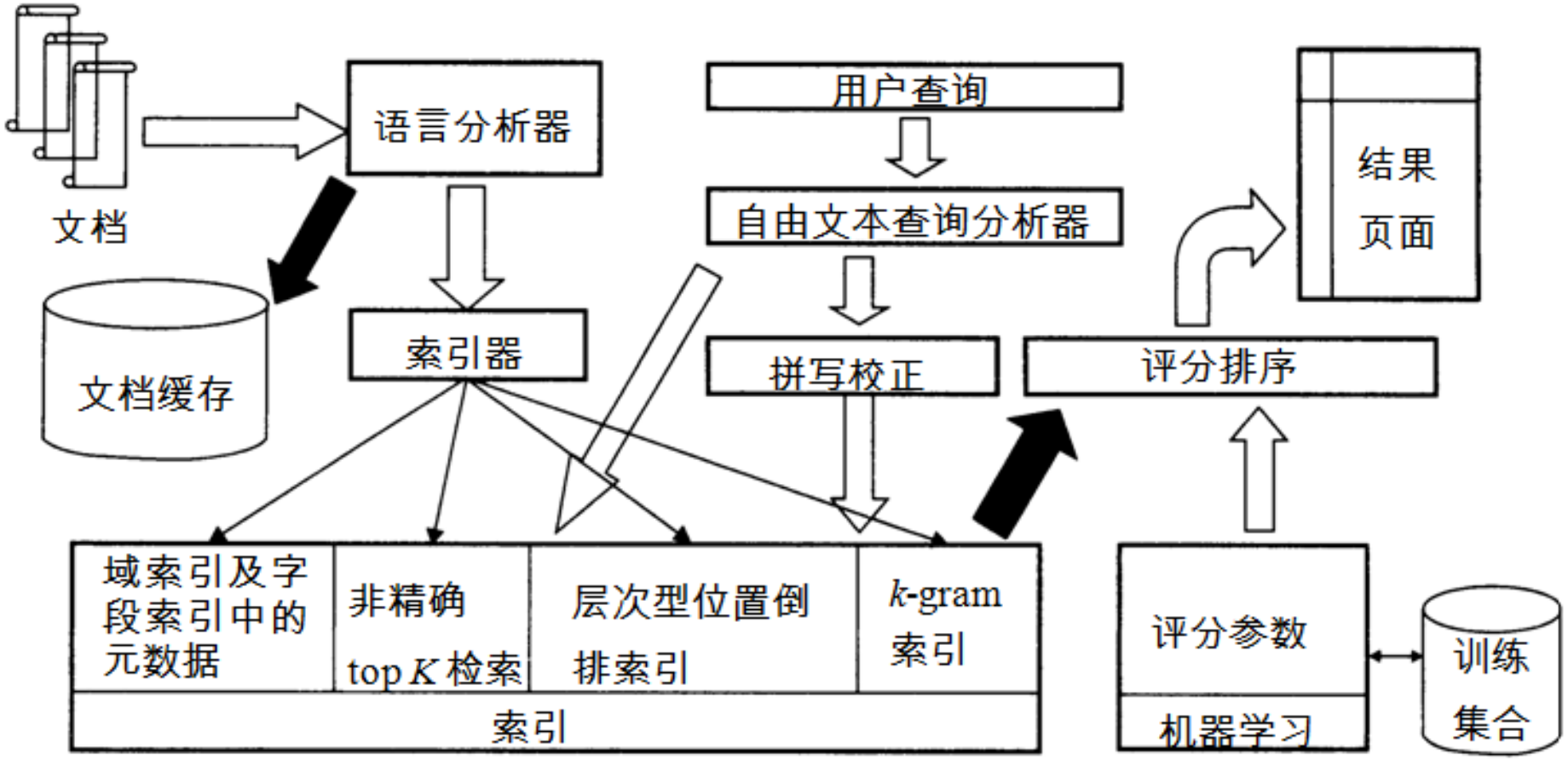
瓶颈的消除

- 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

提纲

- 上一讲回顾
- 结果排序的动机
- 再论余弦相似度
- 结果排序的实现
- 完整的搜索系统

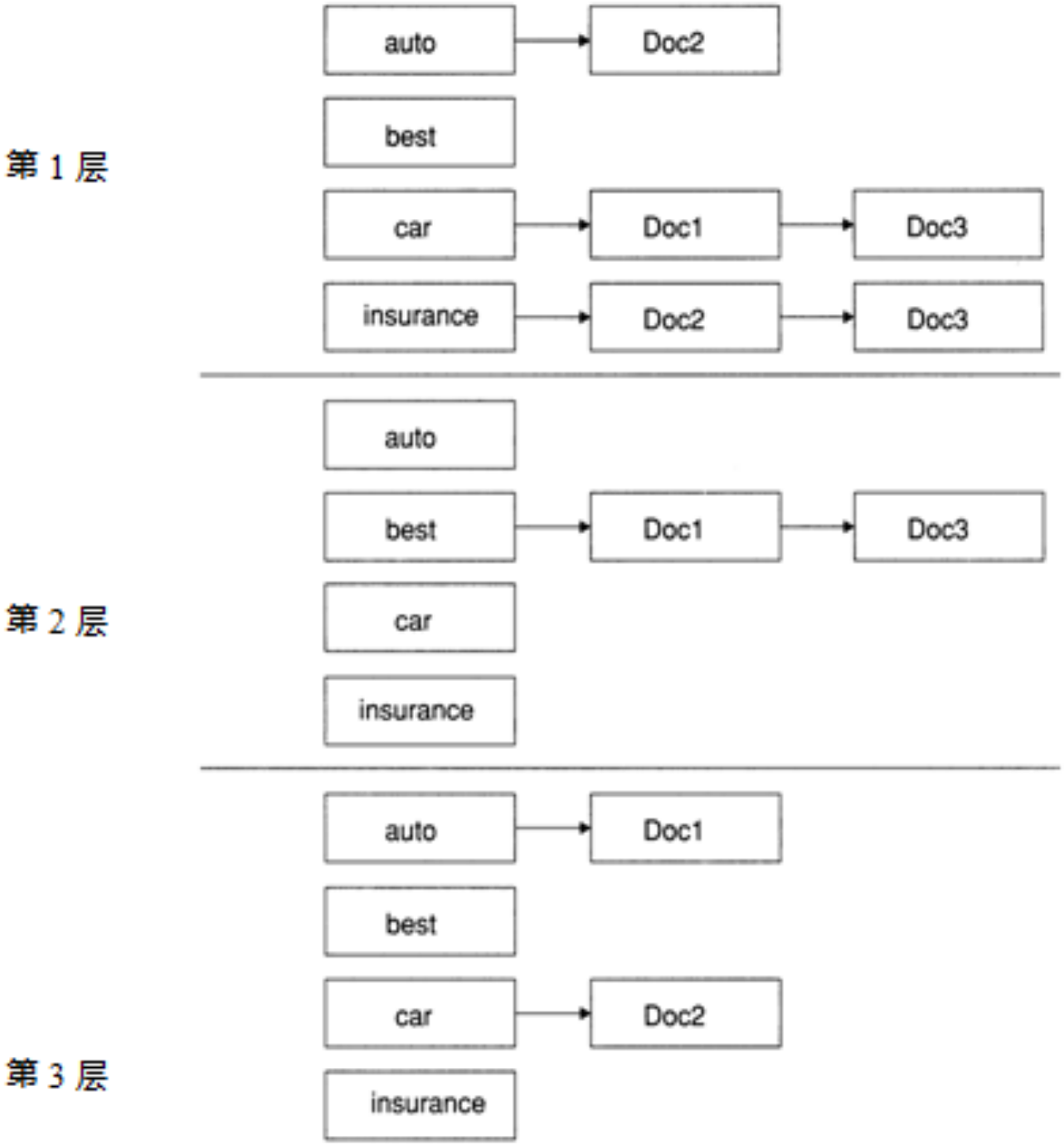
完整的搜索系统示意图


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


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

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