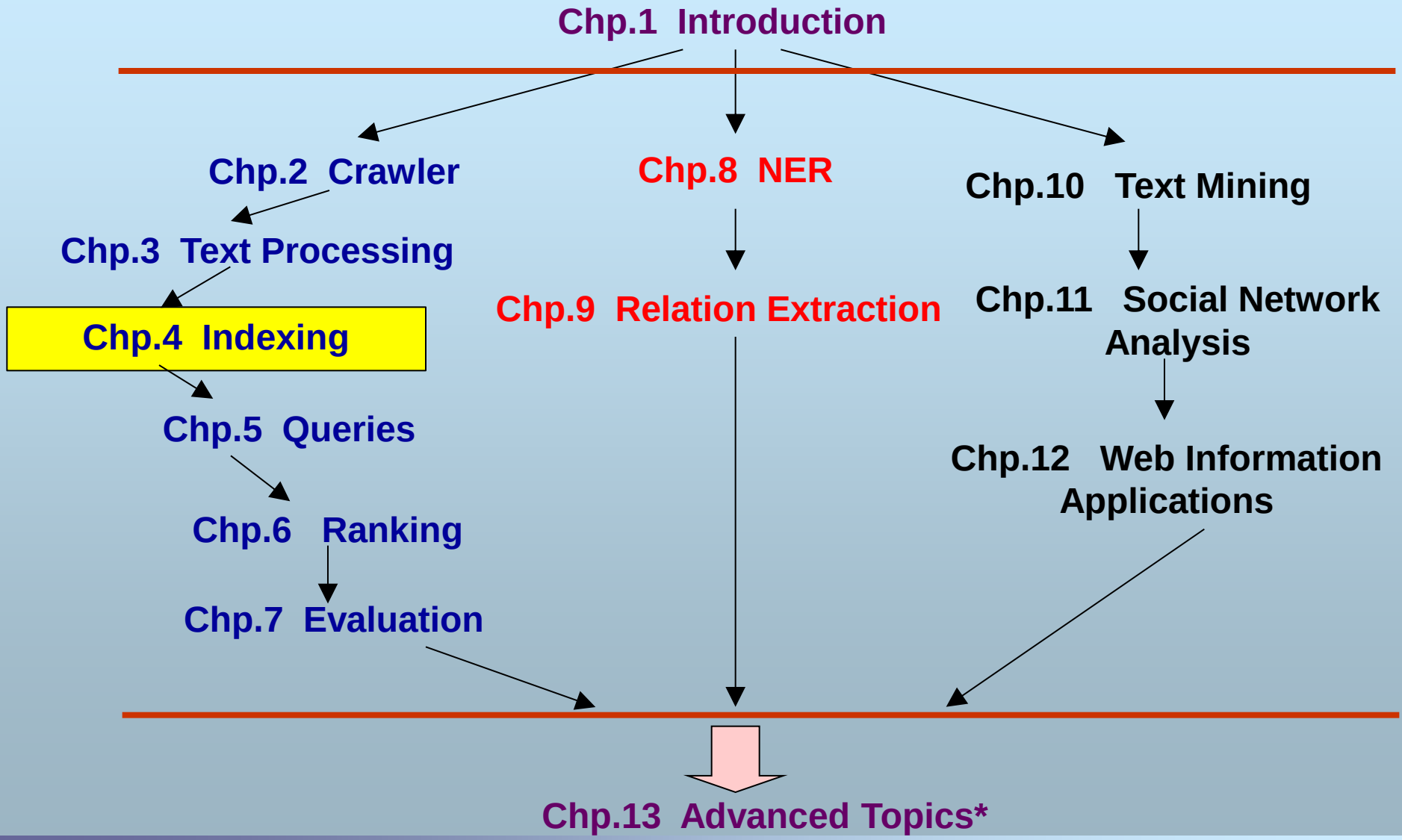
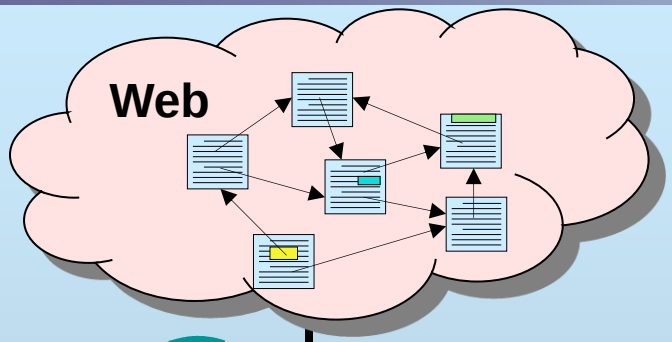
Indexing

金培权（jpq@ustc.edu.cn）

课程知识结构

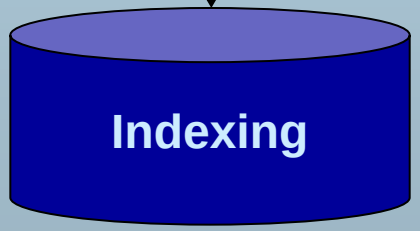

Chp.1 Introduction
Chp.2 Crawler
Chp.3 Text Processing
Chp.4 Indexing
Chp.5 Queries
Chp.6 Ranking
Chp.7 Evaluation
Chp.8 NER
Chp.9 Relation Extraction
Chp.10 Text Mining
Chp.11 Social Network Analysis
Chp.12 Web Information Applications
Chp.13 Advanced Topics*

本章讨论的问题



Crawler/
Spider

Text Processing



预处理后的词项如何组织和存储？

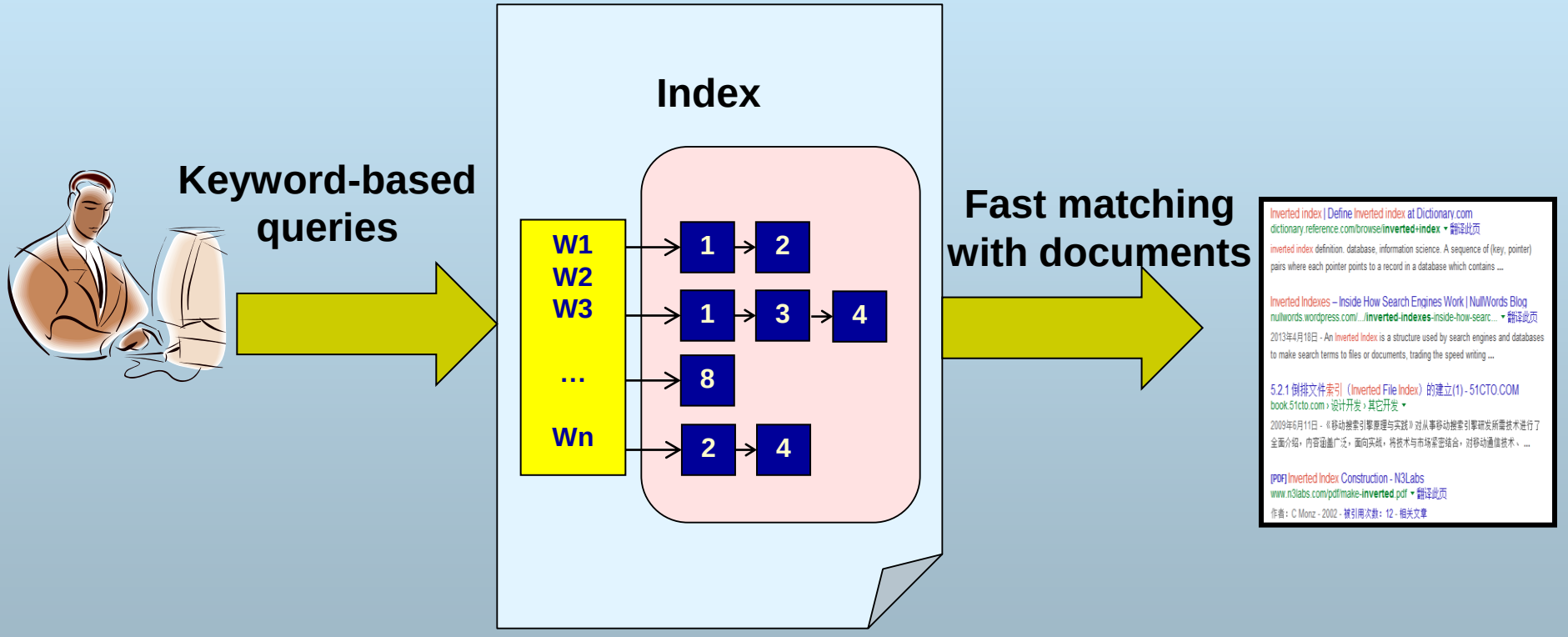
Using an *index* to maintain terms effectively and efficiently

Query/Ranking

Info. Extraction

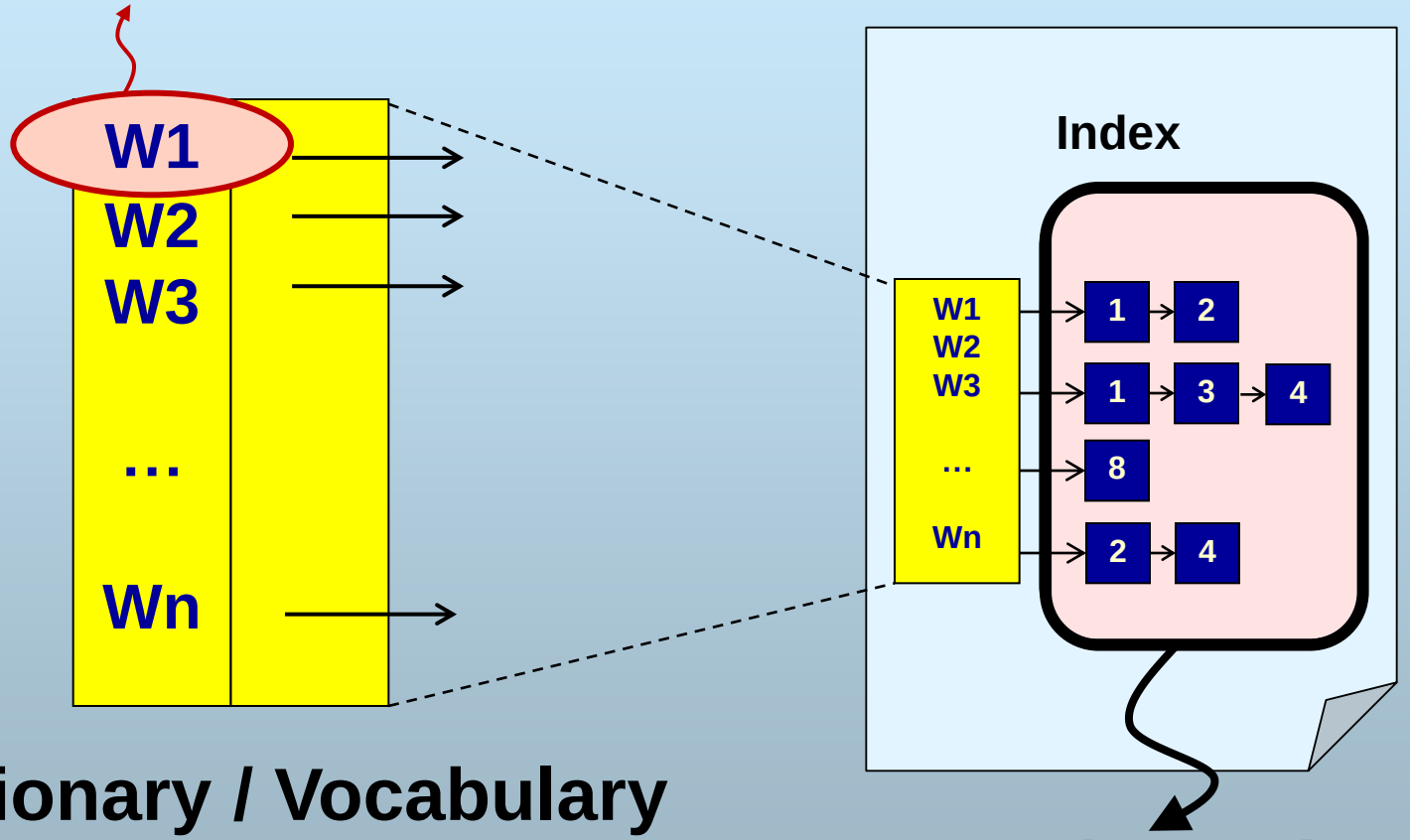
Web Mining

The Goal of Indexing



The Goal of Indexing

Index terms / Search keys in DB



本章主要内容

- 文档分析
- 倒排索引

一、文档分析

中国科学技术大学，[2]，标准简称为**中国科大**，常用简称**科大**或**USTC**，是中国大陆的一所公立研究型大学，[3]学校主体位于安徽省合肥市。

中国科学技术大学隶属于中国科学院，是全国唯一由中国科学院直属管理的全国重点大学。本科生生源和培养质量一直在全国高校中名列前茅。为中国首批7所“211工程”重点建设的大学和首批9所“985工程”重点建设的大学之一[4]；是国家“111计划”和“珠峰计划”重点建设的研究型大学；也是“2011计划”中“量子信息与量子科技前沿协同创新中心”的主要协同单位之一。学校在国际上也享有一定声誉，东亚研究型大学协会和环太平洋大学联盟的成员。是九校联盟（C9）和长三角高校合作联盟的重要成员。中国大学校长联谊会成员。中国科学技术大学微尺度物质科学国家实验室入选海外创新人才基地。英国《泰晤士报高等教育副刊》公布该报2010年世界大学排行榜，中国科学技术大学名列全球第49位，中国大陆第二位，同中国内地北京大学，清华大学共有3所高校进入世界百强。英国《泰晤士报高等教育副刊》发布2011~2012世界大学排行榜，中国科学技术大学排名第192位，次于北京大学和清华大学，位居中国大陆第三。[5]办学目标定位于“ 质量优异、特色鲜明、规模适度、结构合理的一流研究型大学”。

1、索引词项的选择

- 索引词项的选择范围
  - 人工索引—>质量高，但不适用大规模文档数据处理
  - 自动索引
    - 部分索引—> title, abstract, keywords, etc
    - 全文索引—>文档中所有词都参与索引 (SE/IR普遍采用)
- 索引词项的选择原则
  - Index term ≠ word
    - 理想：表达文档内容的语义单位
    - 依赖文本处理技术：stemming、stopwords……

2、用户检索方式

- 最简单的检索方式：布尔检索
  - 指利用 AND, OR 或者 NOT操作符将词项连接起来的查询
    - *信息* AND *检索*
    - *信息* OR *检索*
    - *信息* AND *检索* AND NOT *教材*
  - 在30多年中是最主要的检索工具
  - 当前许多搜索系统仍然使用布尔检索模型
    - 电子邮件、文献编目、Mac OS X Spotlight工具

2、用户检索方式

- 最简单的检索方式：布尔检索

3、检索例子

- 莎士比亚的哪部剧本包含Brutus及Caesar但是不包含Calpurnia?
  - 布尔表达式为 Brutus AND Caesar AND NOT Calpurnia。
- 笨方法：从头到尾扫描所有剧本，对每部剧本判断它是否包含Brutus AND Caesar，同时又不包含Calpurnia
  - 速度超慢 (特别是大型文档集)
  - 处理NOT Calpurnia 并不容易 (需要遍历文档)
  - 不太容易支持其他操作 (e.g., find the word Romans near countrymen)
  - 不支持检索结果的排序 (即只返回较好的结果)

4、另一种选择

Term-Document 关联矩阵 (Incidence Matrix)

Terms \ Docs	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

1 if play contains word,
0 otherwise

Q: Brutus AND Caesar but *NOT Calpurnia*

Incidence Matrix

- 关联矩阵的每一列都是 0/1向量，每个0/1都对应一个词项
- 给定查询Brutus AND Caesar AND NOT Calpurnia
  - 取出三个行向量，并对Calpurnia 的行向量求补，最后按位进行与操作

110100 *AND* 110111 *AND* 101111 = 100100

Answer to the query

- **Antony and Cleopatra**
  - **Act III, Scene ii**

> ***Agrippa*** **[Aside to DOMITIUS ENOBARBUS]: Why, Enobarbus,**
> **When Antony found Julius *Caesar* dead,**
> **He cried almost to roaring; and he wept**
> **When at Philippi he found *Brutus* slain.**

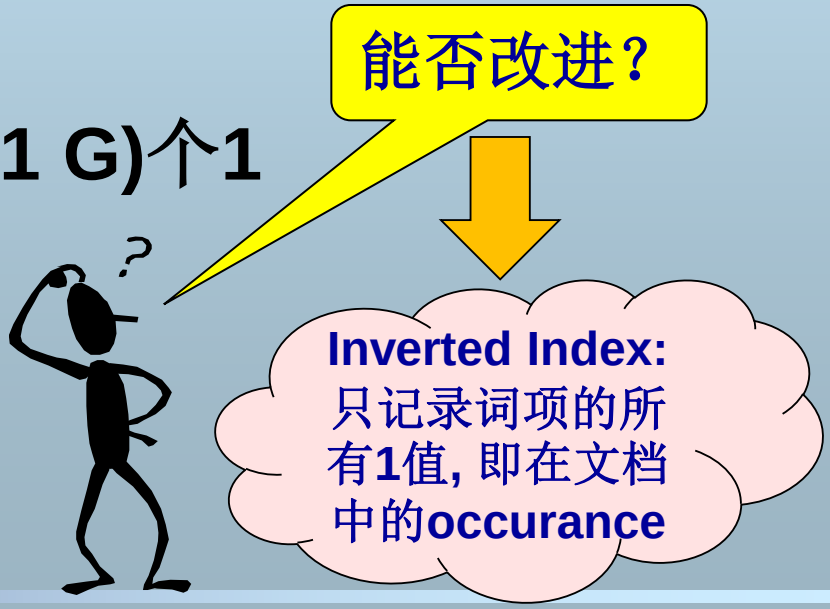
- **Hamlet**
  - **Act III, Scene ii**

> ***Lord Polonius:*** **I did enact Julius *Caesar* I was killed i' the Capitol; *Brutus* killed me.**

Q：Brutus AND Caesar **but** ***NOT Calpurnia***

5、如果文档集很大

- 假定有1 百万篇文档(1 M), 每篇有1000个词(1 K)，假定每个词平均有6个字节
  - 那么所有文档将约占6 GB
- 假定词汇表的大小(即term个数) M = 500 K，则关联矩阵会很大
  - 500 K x 1 M = 500G
- 但矩阵中最多只会有10亿(1 G)个1
  - 1 M * 1 K
  - 高度稀疏(sparse)，1/500



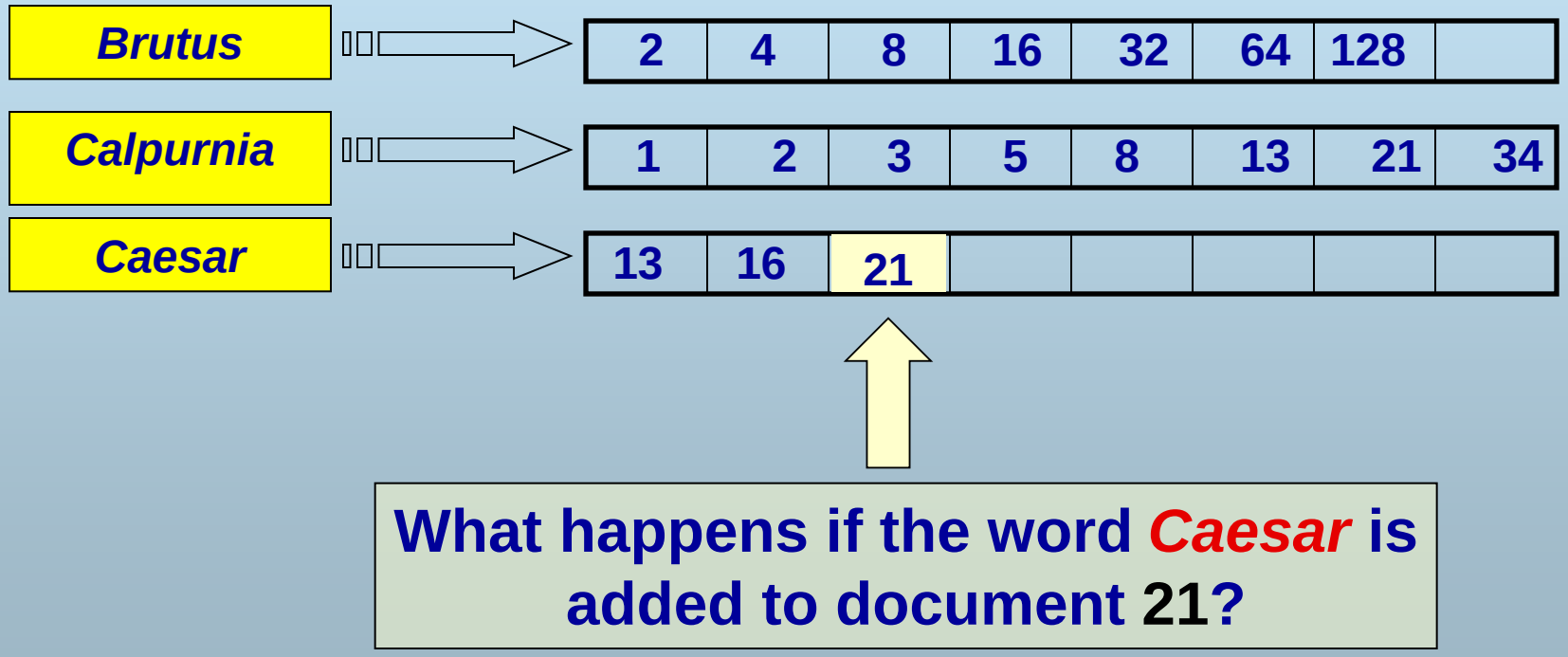
二、倒排索引

- IR中流行的基于词项的文本索引
- 包含两部分结构
  - Vocabulary (Dictionary) : a set of terms
  - Postings List: doc list where the term appeared

Vocabulary		Postings List
term1	→	Document17, Document 45123
		.
		.
termN	→	Document991, Document123001

1、倒排索引示例1

- 莎士比亚的剧本索引



2、倒排索引示例2

Document D1: “yes we got no bananas”

Document D2: “Johnny Appleseed planted apple seeds.”

Document D3: “we like to eat, eat, eat apples and bananas”

Vocabulary		Postings List
yes	→	D1
we	→	D1, D3
got	→	D1
no	→	D1
bananas	→	D1, D3
Johnny	→	D2
Appleseed	→	D2
planted	→	D2
apple	→	D2, D3
seeds	→	D2
like	→	D3
to	→	D3
eat	→	D3
and	→	D3

Query
“apples bananas”:

“apples” → {D2, D3}
“bananas” → {D1, D3}

Whole query gives the *intersection*:

{D2, D3} ^ {D1, D3} = {D3}

3、倒排索引扩展

- 扩展词项在文档中的位置
  - 可以支持短语查询

Document D1: “yes we got no bananas”

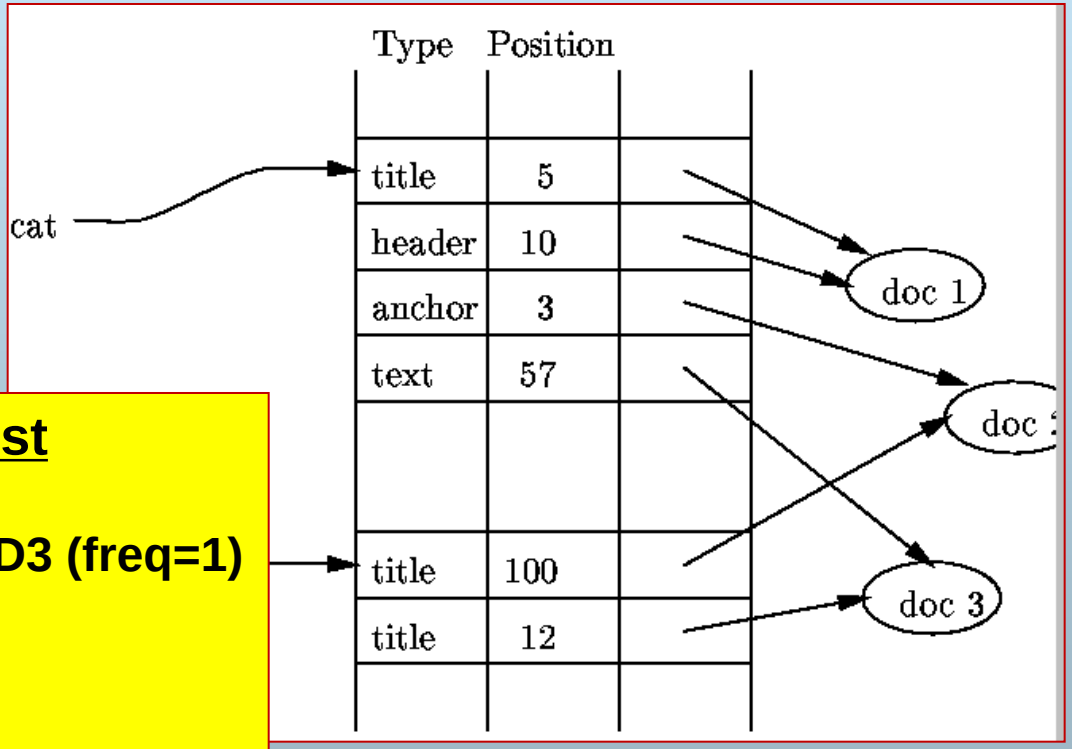
Document D2: “Johnny Appleseed planted apple seeds.”

Document D3: “we like to eat, eat, eat apples and bananas”

Vocabulary		Postings List
yes	→	D1 (+0)
we	→	D1 (+1), D3 (+0)
got	→	D1 (+2)
no	→	D1 (+3)
bananas	→	D1 (+4), D3 (+8)
…		
eat	→	D3 (+3,+4,+5)
…		

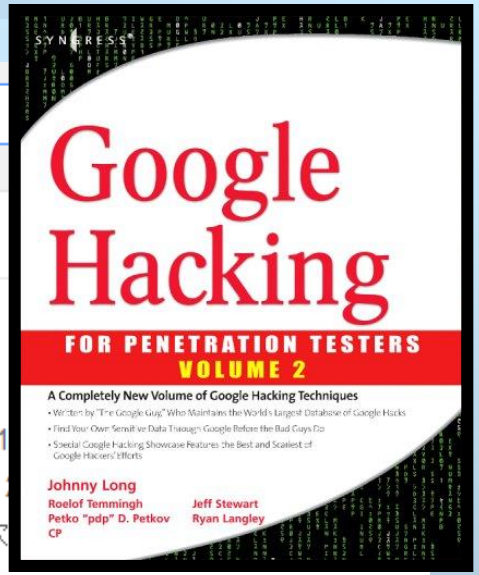
3、倒排索引扩展

- 搜索引擎往往在索引中加入更多的扩展项
  - TF(Term Freq.)、DF(Doc. Freq.)、Term Type
  - 以更好地支持ranking
  - 提高查询效率



Vocabulary	Postings List
yes (docs=1) →	D1 (freq=1)
we (docs=2) →	D1 (freq=1), D3 (freq=1)
…	
eat (docs=1) →	D3 (freq=3)
…	

3、倒排索引扩展



Google moving objects database filetype:pdf site:edu

网页 图片 地图 更多 搜索工具

找到约 45,400 条结果 （用时 0.30 秒）

Google 学术：moving objects database filetype:pdf site:edu

Location management in moving objects databases - Wolfson - 被引用次数：41

... -server architectures for object oriented database ... - DeWitt - 被引用次数：

Moving object detection, tracking and classification for ... - Dedeoglu - 被引用次

[PDF] MoveMine: Mining Moving Object Databases - University of Illinois ...
www.cs.uiuc.edu/~hanj/pdf/sigmod10_zli.pdf ▾ 翻译此页

作者：Z Li - 2010 - 被引用次数：31 - 相关文章

MoveMine: Mining Moving Object Databases ∗. Zhenhui Li‡. Ming Ji‡. Jae-Gil Lee§. Lu-An Tang‡. Yintao Yu‡. Jiawei Han‡. Roland Kays†. ‡ University of ...

[PDF] Spatial and Moving Objects Databases - Department of Computer ...
www.cise.ufl.edu/.../Spatial%20and%20**Moving**%20**Objects**%2... ▾ 翻译此页

Spatial and Moving Objects Databases: St t f th A t d F t. R h Ch ll. State of the Art and Future Research Challenges. Markus Schneider f. University of Florida.

[PDF] Algorithms for Moving Objects Databases - Department of Comput...
www.cise.ufl.edu/~mschneid/Research/papers/CFGNS03CJ.pdf ▾ 翻译此页

4、倒排索引构建

Algorithm:

1. Scan each document, word by word
   - Write *<term, docID>* pair for each word to TempIndex file
2. Sort TempIndex by terms
3. Iterate through sorted TempIndex:
   - merge all entries for the same term into one postings list.

4、倒排索引构建

Algorithm:

1. Scan through each document, word by word
   - Write *<term, docID>* pair for each word to TempIndex file

yes	→ D1
we	→ D1
got	→ D1
no	→ D1
bananas	→ D1

Johnny	→ D2
Appleseed	→ D2
planted	→ D2
apple	→ D2
seeds	→ D2

we	→ D3
like	→ D3
to	→ D3
eat	→ D3
eat	→ D3
eat	→ D3
apples	→ D3
and	→ D3
bananas	→ D3

Document D1: “yes we got no bananas”

Document D2: “Johnny Appleseed planted apple seeds.”

Document D3: “we like to eat, eat, eat apples and bananas”

4、倒排索引构建

Algorithm:

1. Scan through each document, word by word

- Write *<term, docID>* pair for each word to TempIndex file

TempIndex:

yes	→ D1	we	→ D3
we	→ D1	like	→ D3
got	→ D1	to	→ D3
no	→ D1	eat	→ D3
Bananas	→ D1	eat	→ D3
Johnny	→ D2	eat	→ D3
Appleseed	→ D2	apples	→ D3
planted	→ D2	and	→ D3
apple	→ D2	bananas	→ D3
seeds	→ D2		

4、倒排索引构建

Algorithm:

2. Sort TempIndex by terms

TempIndex:

and	→ D3	Johnny	→ D2
apple	→ D2	like	→ D3
apple	→ D3	no	→ D1
Appleseed	→ D2	planted	→ D2
bananas	→ D1	seeds	→ D2
bananas	→ D3	to	→ D3
eat	→ D3	we	→ D1
eat	→ D3	we	→ D3
eat	→ D3	yes	→ D1
got	→ D1		

4、倒排索引构建

Algorithm:

3. Merge postings lists for matching terms

TempIndex:

and	→ D3	Johnny	→ D2
apple	→ D2	like	→ D3
apple	→ D3	no	→ D1
Appleseed	→ D2	planted	→ D2
bananas	→ D1	seeds	→ D2
bananas	→ D3	to	→ D3
eat	→ D3	we	→ D1
eat	→ D3	we	→ D3
eat	→ D3	yes	→ D1
got	→ D1		

4、倒排索引构建

Algorithm:

3. Merge postings lists for matching terms

TempIndex:

and	→ D3	Johnny	→ D2
apple	**→ D2, D3**	like	→ D3
Appleseed	→ D2	no	→ D1
bananas	→ D1	planted	→ D2
bananas	→ D3	seeds	→ D2
eat	→ D3	to	→ D3
eat	→ D3	we	→ D1
eat	→ D3	we	→ D3
got	→ D1	yes	→ D1

4、倒排索引构建

Algorithm:

3. Merge postings lists for matching terms

TempIndex:

and	→ D3	Johnny	→ D2
apple	→ D2, D3	like	→ D3
Appleseed	→ D2	no	→ D1
bananas	**→ D1, D3**	planted	→ D2
eat	→ D3	seeds	→ D2
eat	→ D3	to	→ D3
eat	→ D3	we	→ D1
got	→ D1	we	→ D3
		yes	→ D1

4、倒排索引构建

Algorithm:

3. Merge postings lists for matching terms

TempIndex:

and	→ D3	no	→ D1
apple	→ D2, D3	planted	→ D2
Appleseed	→ D2	seeds	→ D2
bananas	→ D1, D3	to	→ D3
eat	**→ D3**	we	→ D1
got	→ D1	we	→ D3
Johnny	→ D2	yes	→ D1
Like	→ D3		

4、倒排索引构建

Algorithm:

3. Merge postings lists for matching terms

TempIndex:

and	→ D3	no	→ D1
apple	→ D2, D3	planted	→ D2
Appleseed	→ D2	seeds	→ D2
bananas	→ D1, D3	to	→ D3
eat	→ D3	**we**	→ **D1, D3**
got	→ D1	yes	→ D1
Johnny	→ D2		
Like	→ D3		

4、倒排索引构建

Algorithm:

3. Merge postings lists for matching terms

Final Index:

and	→ D3
apple	→ D2, D3
Appleseed	→ D2
bananas	→ D1, D3
eat	→ D3
got	→ D1
Johnny	→ D2
Like	→ D3
no	→ D1
planted	→ D2
seeds	→ D2
to	→ D3
we	→ D1, D3
yes	→ D1

5、倒排索引存储

- 一种方式：Dictionary与Postings List存储在一个文件中
  - 文档规模大时导致索引过大，影响性能
- 另一种常用方式
  - Dictionary与Postings List分别存储为不同的文件，通过页指针关联
    - Postings List文件也可以分布存储
  - 好处：性能
    - Dictionary有可能可以常驻内存，至少可以常驻一部分
    - 可以支持并行、分布查询

6、词汇表存储结构

- 顺序存储
- **Hash table**
- **B+-Tree**
- **Trie树**

顺序存储

- 词汇表的顺序排列
  - 把词汇按照字典顺序排列
  - 词汇的查找采用二分查找
- 优点
  - 实现简单
- 缺点
  - 索引构建的效率一般
    - 对于插入的文档需要反复地调用排序和查找算法
  - 检索的效率一般
    - 二分查找logN的复杂度
    - 能不能变成和词汇数量无关的常数复杂度？

哈希存储

- 词汇表的哈希存储
  - 根据给定的词项，散列成一个整数
  - 用该整数作为词项的访问地址
- 优点
  - 实现简单
  - 检索速度快，理论时间O(1)
- 缺点
  - 当冲突过多时效率会下降
  - 关键在于找到一个好的散列函数
    - 词项的分布不均匀，且跟应用相关

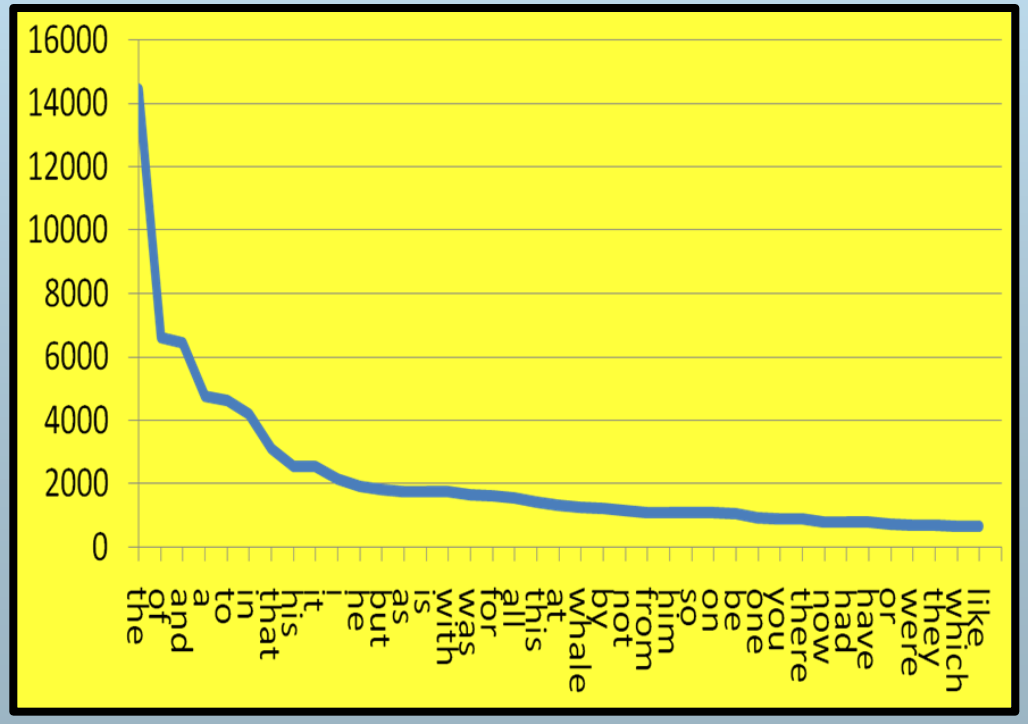
语言学上的Zipf' Law

任意一个term，其频度(freq)的排名(rank)和freq的乘积大致是一个常数

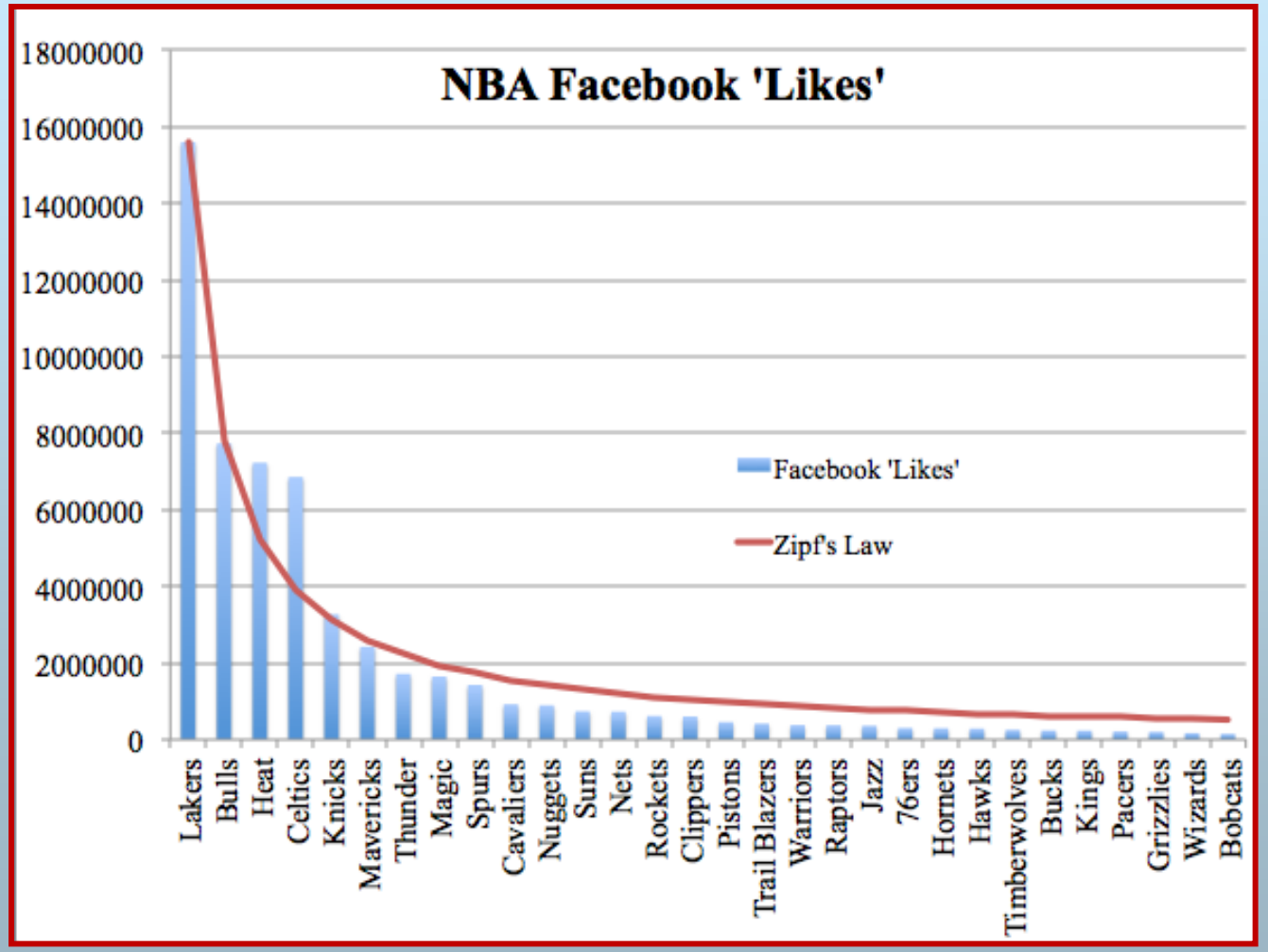
Zipf's law

Rank x Frequency ≈ Constant

Rank	Term	Freq.	Z	Rank	Term	Freq.	Z
1	the	69,971	0.070	6	in	21,341	0.128
2	of	36,411	0.073	7	that	10,595	0.074
3	and	28,852	0.086	8	is	10,099	0.081
4	to	26.149	0.104	9	was	9,816	0.088
5	a	23,237	0.116	10	he	9,543	0.095

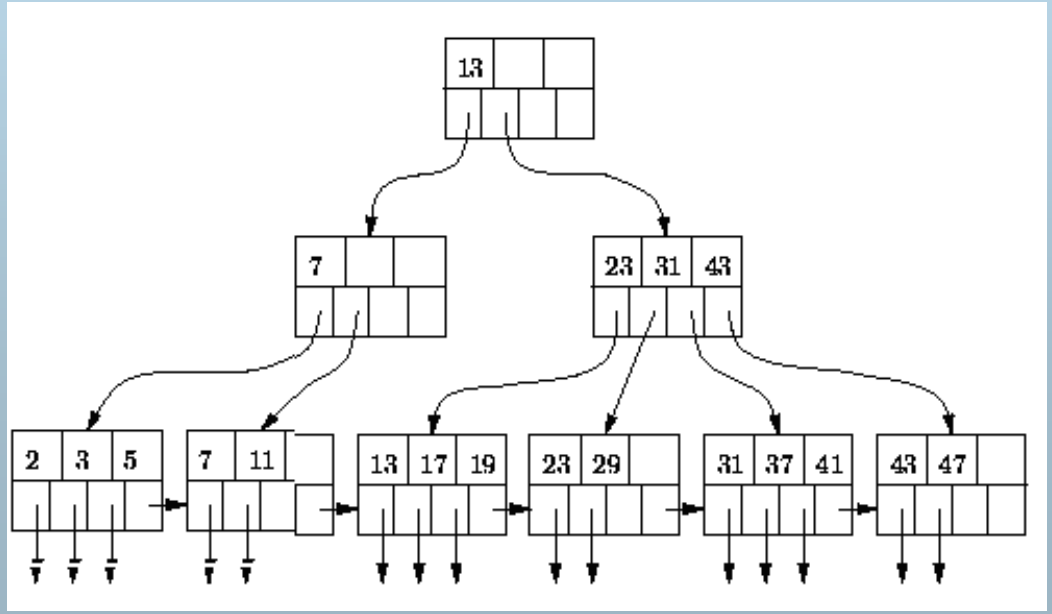


Zipf' Law具有普遍性



B+-Tree存储

- 词汇表的B+-tree存储
  - 多叉平衡有序树
- 优点
  - 性能好而且稳定，查找时间＝树高
  - 适合外存索引
- 缺点
  - 维护代价较高
  - 实现相对复杂



B+-Tree的效率

- 设磁盘块大小8 KB，词项6 B，指针2 B，则一个块可放1024个索引项

层数	索引大小（块数/大小）	索引词项数	查找I/O
1	1/8 KB	1024	1次
2	(1＋1024)/约8 MB	约105万(2^{20})	2次
3	(1+2^{10}+2^{20})/约8 GB	约10.7亿(2^{30})	3次

1次磁盘I/O～＝ 10～30 ms

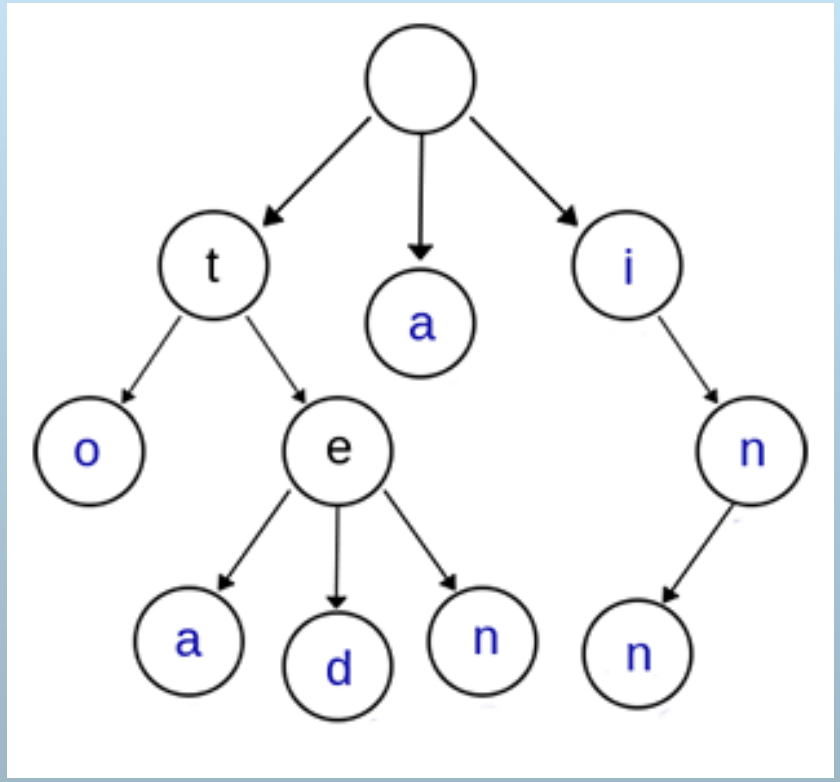
Trie树

- Trie树(from retrieval): 利用字符串的公共前缀来节约存储空间
  - Trie树是一种用于快速检索的多叉树结构
  - Trie树把要查找的关键词看作一个字符序列
  - 根节点不包含字符，除根节点外每一个节点都只包含一个字符
  - 从根节点到某一节点，路径上经过的字符连接起来，为该节点对应的字符串
  - 每个节点的所有子节点包含的字符都不相同
- 查找时间只跟词的长度有关，而于词表中词的个数无关。当词表较大时才能体现出速度优势

Trie树

词典单词：

t、*a*、*i*、*to*、*te*、*in*、*tea*、*ted*、ten、*inn*



Trie树

- 优点
  - 查找效率高，与词表长度无关
  - 索引的插入，合并速度快
- 缺点
  - 所需空间较大
    - 如果是完全m叉树，节点数指数级增长
    - Trie树虽然不是完全m叉树，但所需空间仍然很大，尤其当词项公共前缀较少时
  - 实现较复杂

本章小结

- 文档分析
- 倒排索引
  - 从研究角度，构建搜索系统时一般可参照开源工具，然后加上自己的扩展
  - 最常见的是利用Lucene 建立索引。 Lucene常用的基于Java的搜索服务开发工具包
    - Nutch 是基于Lucene开发的搜索引擎原型系统
  - 另一个流行的Java工具包是Lemur
    - Indri是基于Lemur的搜索引擎的原型系统。