

信息检索导论

An Introduction to Information Retrieval

第2讲 词汇表和倒排记录表

The term vocabulary and postings lists

授课人：李波

中国科学院信息工程研究所/国科大网络空间安全学院

提纲

① 上一讲回顾

② 文档

③ 词项

- 通常做法+非英语处理
- 英语

④ 跳表指针

⑤ 短语查询

提纲

① 上一讲回顾

② 文档

③ 词项

- 通常做法+非英语处理
- 英语

④ 跳表指针

⑤ 短语查询

上一讲回顾

- 什么是信息检索？
- 为什么要学习信息检索？
- 课程情况
- 布尔检索

布尔查询

- 布尔查询: Brutus AND Caesar AND NOT Calpurnia
- 暴力方法(线性扫描)有很多不足: 大规模情况下太慢
- 以空间换查询时间: 事先将词项、文档表示成词项-文档矩阵

词项-文档(term-doc)的0-1关联矩阵

	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

Brutus AND Caesar AND NOT Calpurnia

若某剧本包含某单词，则该位置上为1，否则为0

词项-文档矩阵太大

- 对于某个文档集，大小为6GB，但是词项-矩阵中的元素个数为500G，其中只有1G个元素非零(0.2%非零)，矩阵极度稀疏
- 因此，只需要存储所有的非零元素，于是得到所谓的倒排索引(Inverted Index)
- 原始文档集➡词项-文档矩阵➡倒排索引

构建倒排索引的初体验

对每个词项 t , 保存所有包含 t 的 文档列表

BRUTUS	→	1	2	4	11	31	45	173	174
--------	---	---	---	---	----	----	----	-----	-----

CAESAR	→	1	2	4	5	6	16	57	132	...
--------	---	---	---	---	---	---	----	----	-----	-----

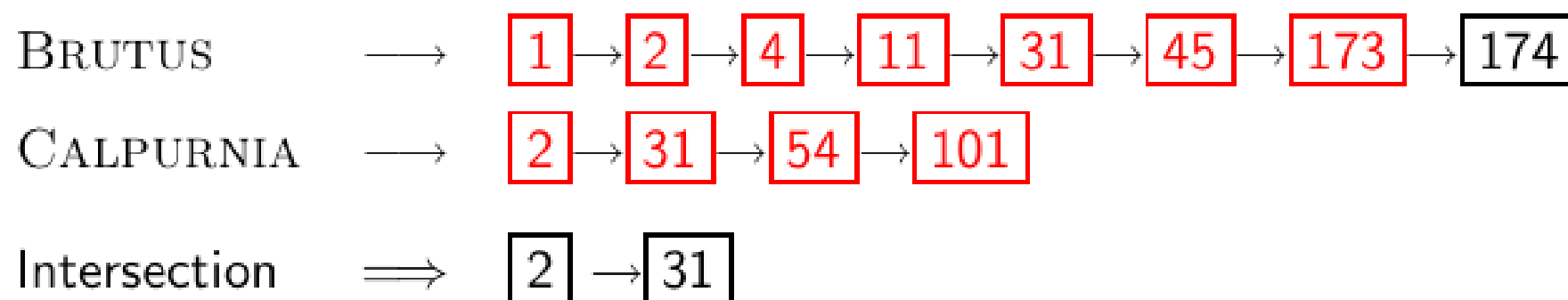
CALPURNIA	→	2	31	54	101
-----------	---	---	----	----	-----

⋮

词典(dictionary)

倒排记录表(postings)

倒排记录表的合并(求交集)

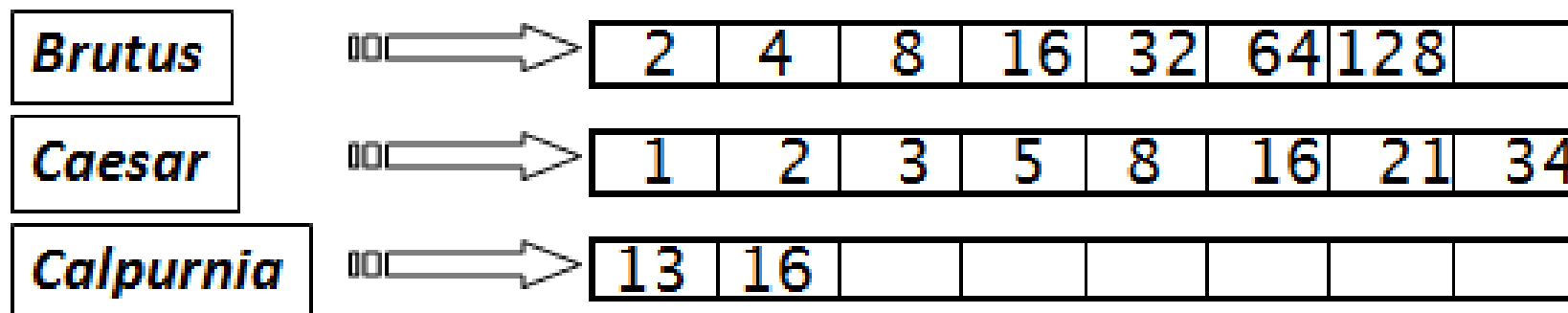


倒排索引构建: 将倒排记录排序

term	docID		term	docID
i	1		ambitious	2
did	1		be	2
enact	1		brutus	1
julius	1		brutus	2
caesar	1		capitol	1
i	1		caesar	1
was	1		caesar	2
killed	1		caesar	2
i'	1		did	1
the	1		enact	1
capitol	1		hath	1
brutus	1		i	1
killed	1		i	1
me	1	⇒	i'	1
so	2		it	2
let	2		julius	1
it	2		killed	1
be	2		killed	1
with	2		let	2
caesar	2		me	1
the	2		noble	2
noble	2		so	2
brutus	2		the	1
hath	2		the	2
told	2		told	2
you	2		you	2
caesar	2		was	1
was	2		was	2
ambitious	2		with	2

查询优化

- 按照表从小到大(即df从小到大)的顺序进行处理:
 - 每次从最小的开始合并



相当于处理查询 (*Calpurnia AND Brutus*) *AND Caesar*.

更通用的优化策略

- e.g., (madding OR crowd) AND (ignoble OR strife)
 - 每个布尔表达式都能转换成上述形式(合取范式)
- 获得每个词项的df
- (保守)通过将词项的df相加, 估计每个OR表达式对应的倒排记录表的大小
- 按照上述估计从小到大依次处理每个OR表达式.

一个布尔搜索引擎Westlaw: 例子

需求：联邦侵权索赔法的诉讼时效是什么？(What is the **statute of limitations** in cases involving the **federal tort claims act**?)

查询： **LIMIT! /3 STATUTE ACT /S FEDERAL /2 TORT /3 CLAIM**

/3 = within 3 words, /S = in same sentence,
Space = disjunction

Google中是否使用布尔模型？

- Google默认是与(AND)操作，输入查询 $[w_1 w_2 \dots w_n]$ 意味着 $w_1 \text{ AND } w_2 \text{ AND } \dots \text{ AND } w_n$
- 当返回文档不包含某个词 w_i 时，*可能是如下情形*:
 - 指向该页面的锚文本包含 w_i
 - 页面包含 w_i 的变形(词形变换，拼写校对，同义等等)
 - 长查询 (n较大)
 - 布尔表达式返回的结果少
- 简单的布尔检索 vs. 结果的排序
 - 简单的布尔检索只返回匹配上的文档，不考虑结果顺序
 - Google和其他大部分精心设计的布尔引擎均对结果进行排序，以使好的结果排在差的结果的前面

本讲的内容

- 索引构建过程(特别是预处理)
- 如何对索引文档进行处理来得到词典
 - 理解文档(document)的概念
 - 词项生成, 理解词项(term)的概念
 - 理解词条(token)、词条化(Tokenization)的概念
- 倒排记录表
 - 更快的合并算法: 跳表法(skip list)
 - 短语查询的处理及带位置信息的倒排索引

提纲

① 上一讲回顾

② 文档

③ 词项

- 通常做法+非英语处理
- 英语

④ 跳表指针

⑤ 短语查询

回顾倒排索引构建

待索引文档



Friends, Romans, countrymen.

Tokenizer

词条化工具

词条流

Friends

Romans

Countrymen

Linguistic modules

语言分析工具

修改后的词条

friend

roman

countryman

Indexer

倒排索引

friend

roman

countryman



2 → 4 →

1 → 2 →

13 → 16

回顾倒排索引构建

- 前提
 - 我们知道“待索引的文档”是什么
 - 文档内容可以“machine-read”
- 然而
 - 实际情况要复杂得多

文档分析

- 文档格式处理
 - pdf/word/excel/html?
- 文档语言识别
 - English/中文/日文?
- 文档编码识别
 - UTF-8/zh_CN.***

以上问题都可以看成分类问题，基于后面章节的分类方法可以处理（Chap 13）
但是实际中，常常采用启发式方法.....

多格式/语言并存

- 待索引文档集可能同时包含多种语言的文档
 - 在同一索引中词汇表中包含来自多个语言的词项
- 有时文档或者其部件中包含多种语言/格式
 - 法语邮件中带一个德语的pdf格式附件
- 如何确定索引的单位？
 - 文件为单位？
 - 邮件为单位？
 - 如果邮件带有5个附件，怎么办？
 - 一组文件？（比如采用html格式写的某个PPT文档）
- **结论：回答“文档是什么”的问题并非无关重要，需要设计过程中做出判断**

提纲

① 上一讲回顾

② 文档

③ 词项

- 通常做法+非英语处理
- 英语

④ 跳表指针

⑤ 短语查询

词条和词项

TOKENS AND TERMS

定义

- Word词
 - 文本中用分隔符分开的字符串（在英文中的定义）
- Token词条
 - 文档中出现的词或词项的实例
- Term词项
 - 一个“规范化”的词语（词形变换、拼写变换）
 - 相同类的词
- Type词类
 - 多个词条构成的等价类(equivalence class)集合，通常和词项含义相同

回顾倒排索引构建

待索引文档



Friends, Romans, countrymen.

Tokenizer

词条化工具

词条流

Friends Romans Countrymen

Linguistic modules

语言分析工具

修改后的词条

friend roman countryman

Indexer

倒排索引

friend
friend
roman
countryman

roman

countryman
2 → 4
1 → 2
13 → 16

词条化(Tokenization)

- Tokenization词条化
 - 将文档分割成词条的过程
- 输入：

Friends, Romans, countrymen.
- 输出: 词条(Token)

Friends

Romans

Countrymen
- 词条在经过进一步处理之后将放入倒排索引中的词典中
 - 后面会讲
- 词条化中的问题-词条如何界定?

词条化(Tokenization)练习

- In June, the dog likes to chase the cat in the barn.
 - 有多少词条? 有多少词项/词类?
- Mr. O'Neill thinks that the boys' stories about Finland's capital aren't amusing.
 - 进行分词, 有多少词条?

词条化

- 词条是一个词还是两个或多个词？
 - Finland's capital →
 - Finland? Finlands? Finland's?
 - Hewlett-Packard → 是看成Hewlett 和 Packard 两个词条还是一个词条？
 - state-of-the-art:
 - co-education
 - lowercase, lower-case, lower case ?
 - San Francisco: 到底是一个还是两个词条？
 - 如何判断是一个词条？

词条化中数字的处理

- 3/20/91 Mar. 12, 1991 20/3/91
- 55 B.C.
- B-52 【B-52轰炸机，美国的一种轰炸机】
- PGP 密钥：324a3df234cb23e 【PGP是一个基于RSA公匙加密体系的邮件加密软件】
- (800) 234-2333

- 早期的IR系统可能不索引数字
 - 但是数字却常常很有用：比如在Web上查找错误代码
 - (一种处理方法是采用n-gram: 见第三讲)
- 元数据是分开还是一起索引
 - 创建日期、格式等等

词条归一化(Normalization)成词项

- Normalization归一化
 - 将不完全一致的词条规范为一个等价类（词项）的过程
 - 归一化的结果就是词项，而词项就是最终要索引的对象
- 需要将文档和查询中的词归一化成同一形式：
 - U.S.A. 和 USA
- 可以采用隐式规则的方法来将多个词条归一成同一词项，比如
 - 剔除句点
 - U.S.A., USA \ USA
 - 剔除连接符
 - anti-discriminatory, antidiscriminatory \ antidiscriminatory

词条归一化(Normalization)成词项

- 除了前面互换方式(即能够归一化成同一词项的词条之间完全平等, 可以互换)之外, 还可以使用非对称扩展 (asymmetric expansion)方法, 来实现词条到词项的映射
 - window → window, windows
 - windows → Windows, windows
 - Windows (no expansion)
- 该方法功能更强, 但实现代价更大
- 思考: 为什么倾向于不将window, Window, windows, Windows归一化为同一类别?

归一化中的语言问题

- 重音符: 如法语中 *résumé* vs. *resume*.
- 日耳曼语系中的元音变化: 如德语中的 *Tuebingen* vs. *Tübingen* 【德国地名】
 - 应该是一致的
- 最重要的准则
 - 用户在输入查询时遇到这些词如何输入?
- 即使在有重音符号的语言中, 用户也往往不输入这些符号
 - 常常归一化成不带重音符号的形式
 - Tuebingen, Tübingen, Tubingen \ Tubingen

归一化中的语言问题

- 时间格式
 - 7月30日 vs. 7/30
 - 日语中用假名或者汉字表示日期
- 归一化可能与语言相关，因此必须要做语言识别

是德语的“mit”吗？

PETER WILL NICHT MIT

Morgen will ich in MIT

He got his PhD from MIT

- 另外，谨记要将文档和查询中的同义词归一化成同一形式

语言问题：法语和德语

■ 法语

- L'ensemble 【全部】 → 到底是一个还是两个词条？
 - L ? L' ? Le ?
 - 但是常常希望 l'ensemble 能和un ensemble 【一组】匹配
 - 至少在2003年以前，Google没有这样处理
 - 国际化问题！

■ 德语中复合名词连写

- Lebensversicherungsgesellschaftsangestellter
- 'life insurance company employee' 【人寿保险公司员工】
- 德语检索系统往往要使用一个复合词拆分的模块，而且该模块对检索结果的提高有很大帮助(可以提高15%)

语言问题：阿拉伯文

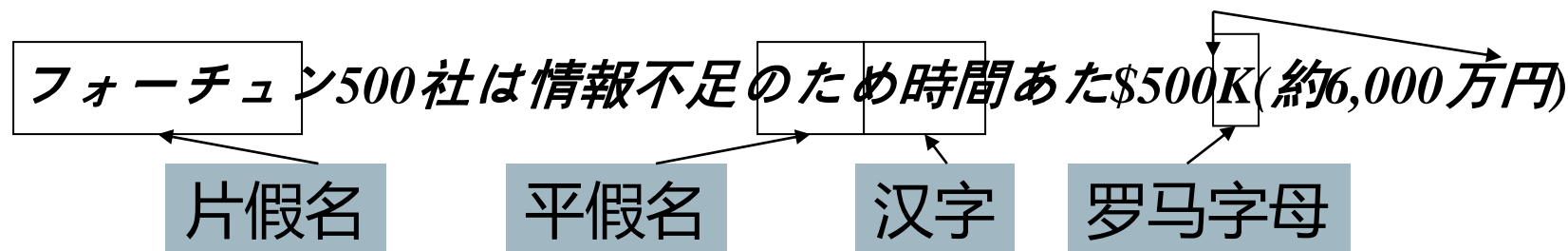
- 阿拉伯文 (或希伯来文) 通常从右到左书写，但是某些部分(如数字)是从左到右书写
- 词之间是分开的，但是单词中的字母形式会构成复杂的连接方式

استقلت الجزائر في سنة 1962 بعد 132 عاما من الاحتلال الفرنسي.

- $\leftarrow \rightarrow \leftarrow \rightarrow$ ← 开始
- ‘Algeria achieved its independence in 1962 after 132 years of French occupation.’
- 在Unicode编码方式下，双向书写将不再是问题

语言问题：中文和日文

- 中文和日文词之间没有间隔：
 - 莎拉波娃现在居住在美国东南部的佛罗里达。
 - 分词结果无法保证百分百正确，“和尚”
- 日文中可以同时使用多种类型的字母表
 - 日期/数字可以采用不同的格式



而终端用户可能完全用平假名方式输入查询！

中文分词(Chinese Word Segmentation)

- 对于中文，分词的作用实际上是要找出一个个的索引单位
- 例子：李明天天都准时上班
- 索引单位
 - 字：李明天天都准时上班
 - 索引量太大，查全率百分百，但是查准率低，比如查“明天”这句话也会出来
 - 词：李明天天都准时上班
 - 索引量大大降低，查准率较高，查全率不是百分百，而且还会受分词错误的影响，比如上面可能会切分成：李明天天都准时上班，还有：他和服务人员照相
 - 字词混合方式/k-gram/多k-gram混合
 - 一般原则，没把握的情况下细粒度优先

中文分词

- 中文分词是很多中文文本处理的第一步
 - 我国科学家近日研制出一套水下反恐监控系统 →
我国 科学家 近日 研制 出 一 套 水下 反恐 监控 系统
- 分词方法
 - 基于是否使用词典：
 - 基于词典的方法：给出一部词典，根据这部词典进行匹配
 - 无词典的方法：不需要词典，根据某种人工构词规则或者统计规则从字生成词。
 - 规则或者统计方法：
 - 基于规则的方法：通过某种判定规则，确定是否为词
 - 统计方法：基于语料库统计+机器学习

中文分词

- 正向最大匹配(基于词典的方法)

0 1 2 3 4 5 6

他	说	的	确	实	在	理
---	---	---	---	---	---	---

指针位置	剩余词串	首字	最大匹配词条
0	他说的确实在理	他	他
1	说的确实在理	说	说
2	的确实在理	的	的确
4	实在理	实	实在
6	理	理	理

中文分词

- 逆向最大匹配(基于词典的方法)

0 1 2 3 4 5 6

他	说	的	确	实	在	理
---	---	---	---	---	---	---

指针位置	剩余词串	尾字	最大匹配词条
6	他说的确实在理	理	在理
4	他说的确实	实	确实
2	他说的	的	的
1	他说	说	说
0	他	他	他

中文分词

- 分词中遇到的两大难题：
 - 未登录词问题(Out of Vocabulary, OOV): 出现词典中没有的词, 如: 人名、地名、机构名、一些新词等等
 - 歧义问题(Ambiguition): 同一句子有多种可能的分词结果
 - 交叉性歧义: 我们小组合成氢气 ➔ 我们/小组/合成/氢气 或 我们/小/组合/成/氢气
 - 组合性歧义: 他/从/马/上/下/来; 我/马上/就/来/了

中文分词

- 解决歧义和未登录词识别的基本方法:
 - 规则方法：分词过程中或者分词结束后根据规则进行处理；
 - 统计方法：分词过程中或者分词结束后根据统计训练信息进行处理。
 - 规则+统计

中文分词和检索

- 以下是当前某些研究的结论或猜测，仅供参考
- 并非分词精度高一定检索精度高
 - 评价标准不同
 - 分词规范问题：鸡蛋、鸭蛋、鹌鹑蛋.....
 - 目标不同
- 检索中的分词：
 - 查询和文档切分采用一致的分词系统
 - 速度快
 - 倾向细粒度，保证召回率
 - 多粒度并存
- 搜索引擎中的分词方法
 - 猜想：大词典+统计+启发式规则

提纲

① 上一讲回顾

② 文档

③ 词项

- 通常做法+非英语处理
- 英语

④ 跳表指针

⑤ 短语查询

大小写问题

- 可以将所有字母转换成小写形式
 - 例外: 句中的大写单词?
 - e.g., General Motors (GM, 通用公司)
 - Fed (美联储)vs. fed(饲养)
 - SAIL (印度钢铁管理局) vs. sail(航行)
 - 通常情况下将所有字母转成小写是一种很合适的方式, 因为用户倾向于用小写方式输入
- Google的例子:
 - 查询 C.A.T.
 - 排名第一的结果是“cat”而不是 Caterpillar Inc.



停用词

- 根据停用词表(stop list), 将那些最常见的词从词典中去掉。比如直观上可以去掉:
 - 一般不包含语义信息的词: the, a, and, to, be
 - 汉语中的 “的”、“得”、“地” 等等。
 - 这些词都是高频词: 前30个词就占了 ~30% 的倒排记录表空间
- 现代信息检索系统中倾向于不去掉停用词:
 - 在保留停用词的情况下, 采用良好的压缩技术(第五章)后, 停用词所占用的空间可以大大压缩, 最终它们在整个倒排记录表中所占的空间比例很小
 - 采用良好的查询优化技术(第七章)基本不会增加查询处理的开销
 - 所谓的停用词并不一定没用, 比如: 短语查询: “King of Denmark”、歌曲名或者台词等等: “Let it be”, “To be or not to be”、“关系型” 查询 “flights to London”

同义词词典及同音词的处理

- 同义词的处理
 - E.g., 手动建立词典, 记录这些词对
 - car = automobile color = colour
 - 利用上述词典进行索引
 - 当文档包含 automobile时, 利用car-automobile进行索引
 - 或者对查询进行扩展
 - 当查询包含 automobile时, 同时也查car
- 拼写错误的处理(如同音词: Clinton→Klinten)
 - 一种解决方法是Soundex方法, 基于发音建立词之间的关系(Soundex方法将在第三章介绍)

词形归并(Lemmatization)

- 将单词的屈折变体形式还原为原形
- 例子:
 - am, are, is → be
 - car, cars, car's, cars' → car
 - the boy's cars are different colors → the boy car be different color
- 词形归并意味中将单词的变形形式“适当”还原成一般词典中的单词形式
 - found → find? found?

词干还原 (Stemming)

- 将词项归约(reduce)成其词干(stem), 然后再索引
- “词干还原”意味着词缀的截除
 - 与语言相关
 - 比如, 将 automate(s), automatic, automation都还原成 automat

for example compressed and compression are both accepted as equivalent to compress.



for exampl compress and
compress ar both accept
as equival to compress

Porter算法

- 英语词干还原中最常用的算法
 - 结果表明该方法不差于其他的词干还原方法
- 一些规定+ 5 步骤的归约过程
 - 这些步骤有先后顺序
 - 每一步都包含一系列命令，例如，如果剩余长度大于1，则删除末尾的ement，replacement→replac，cement→cement
- 一些约定，比如：选择可应用规则组中包含最长后缀的规则
 - SSES →SS caresses →caress
 - S → cats →cat

Porter中的典型规则

- sses → ss
- ies → i
- ational → ate
- tional → tion

- 规则适用条件的表达
 - (m>1) EMENT →
 - replacement → replac
 - cement → cement

Martin Porter

- (应该是)英国人, (应该是)剑桥大学
- 2000年度 Tony Kent Strix award得主
 - 信息检索领域另一个著名的奖项
- Porter's stemmer, 有很多语言的版本
- Snowball 工具, 支持多种语言的 stemming(法语、德语、葡萄牙语、西班牙语挪威语等等)



其他词干还原工具(stemmer)

- Lovins: <http://www.comp.lancs.ac.uk/computing/research/stemming/general/lovins.htm>
 - 单遍扫描，最长词缀剔除
 - 大概 250条规则
- Paice stemmer

三个工具的比较

- **输入文本**: Such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation
- **Porter stemmer**: such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation
- **Lovins stemmer**: such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation
- **Paice stemmer**: such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation

词干还原是否能提高检索效果？


- 全部基于词形分析 – 对于检索来说最多只能提供一定的帮助(at most modest benefits for retrieval)
- 词干还原及其它归一化工作对检索的帮助
 - 英语： 结果要一分为二，对某些查询来说提高了召回率，但是对另外一些查询来说降低了正确率
 - 比如，[sightseeing tour san francisco]查询
 - 比如, operate operating operates operation operative operatives operational → oper, 对[operational AND research], [operating AND system], [operative AND dentistry] 等查询将带来负面影响
- 对西班牙语、德语、芬兰语等语言非常有用
 - 其中对于芬兰语有30% 的性能提高!

语言特性

- 上述很多转换处理具体实现时
 - 都与语言本身有关，并且
 - 常常和具体应用有关
- 上述过程可以插件方式植入索引过程
- 存在很多开源和商业插件可用

词典入口示意图

<i>ensemble.french</i>
<i>時間.japanese</i>
<i>MIT.english</i>
<i>mit.german</i>
<i>guaranteed.english</i>
<i>entries.english</i>
<i>sometimes.english</i>
<i>tokenization.english</i>



可以按(或不按)语言分组, 后面还会讲到

小结

文本

We love you because of what
you do for us



词条化工具

词条

We love you because of what
you do for us (10个词条)



去停用词、归一化 (词根还原、词形归并、
同义词)、....

词项

We love you because of what
you do for us (5个词项)

很多处理可做可不做，可这样做也可以那样做，取决于应用需求！比如停用词去除过程。

思考题

■ 谷歌做了哪些预处理？

- 停用词
- 归一化
- 词条化
- 大小写转换
- 词干还原
- 非拉丁字符
- 变音/重音符号
- 复合词
- 数值
-

提纲

① 上一讲回顾

② 文档

③ 词项

- 通常做法+非英语处理
- 英语

④ 跳表指针

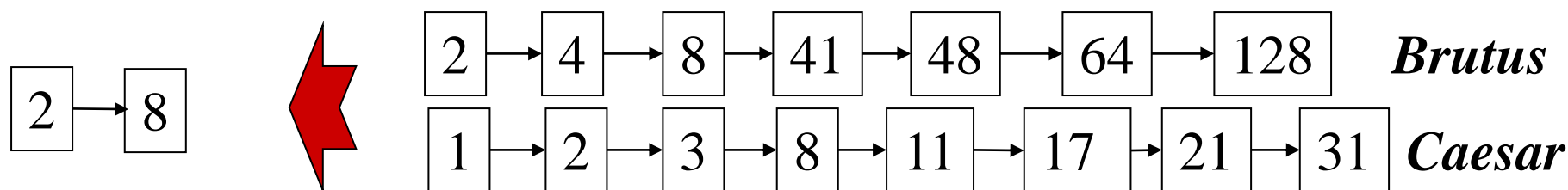
⑤ 短语查询

快速倒排表合并—跳表法

FASTER POSTINGS MERGES: SKIP POINTERS/SKIP LISTS

基本合并算法的回顾

- 两个指针，同步扫描，线性时间

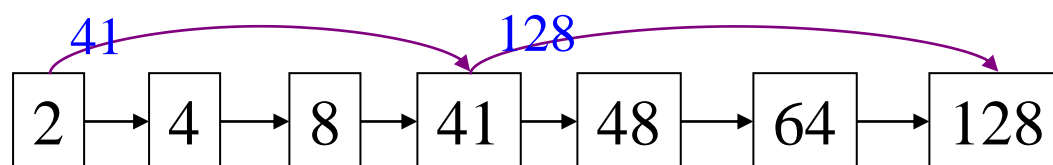


两个表长度为 m 和 n 的话，上述合并时间复杂度为 $O(m+n)$

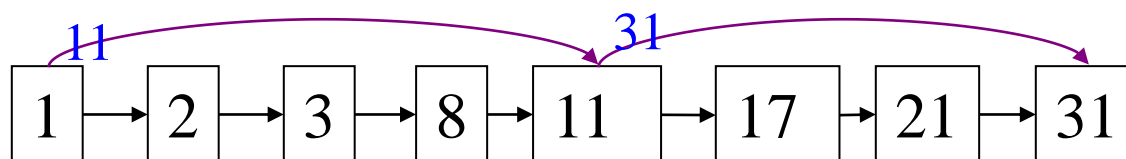
能否做得更好？答案是可以(如果索引不常变化的话)

索引构建时为倒排记录表增加跳表指针

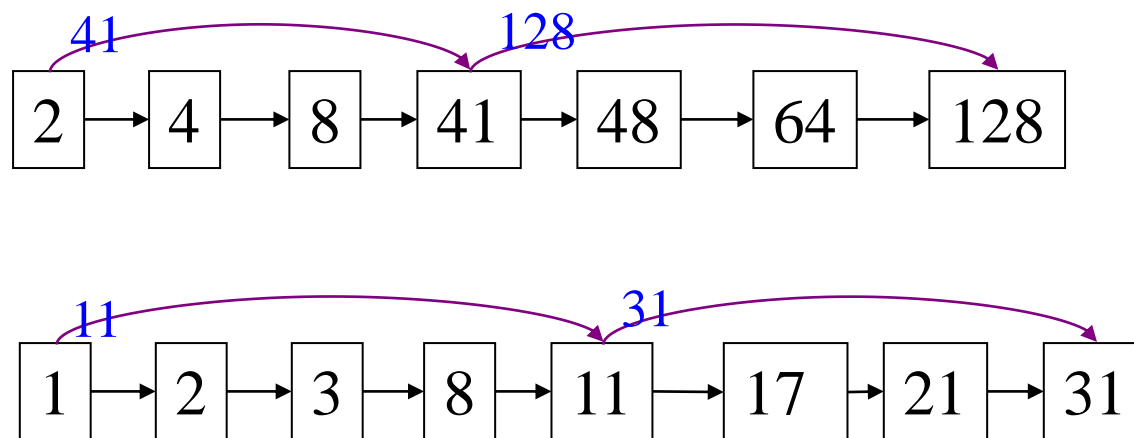
- 为什么可以加快速度？
 - 可以跳过那些不可能的检索结果



- 如何做?也就是在什么地方加跳表指针?



基于跳表指针的查询处理



假定匹配到上下的指针都指向8，接下来两个指针都向下移动一位。

比较41和11，11小

此时看11上面的跳表指针，指向31，31仍然比41小，于是下指针可以直接跳过中间的11、17、21、31

基于跳表指针的倒排记录合并算法

INTERSECTWITHSKIPS(p_1, p_2)

```

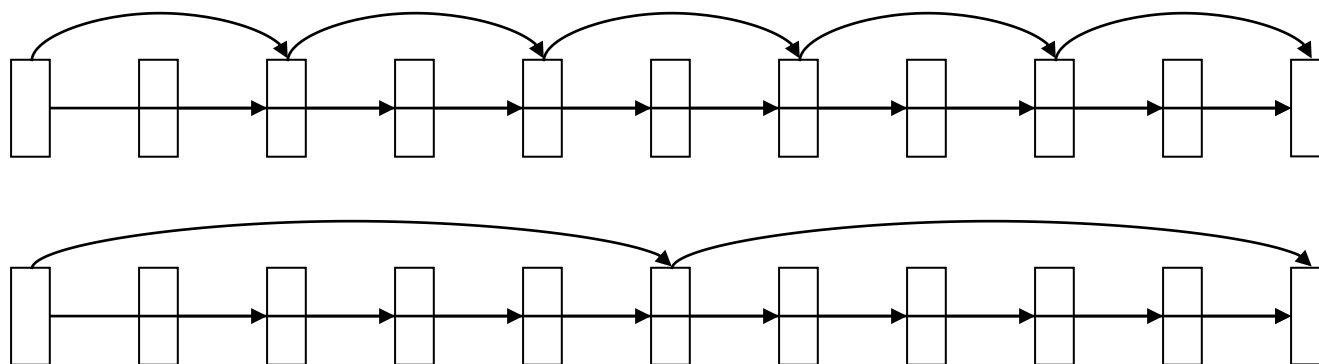
1   $answer \leftarrow \langle \rangle$ 
2  while  $p_1 \neq \text{NIL}$  and  $p_2 \neq \text{NIL}$ 
3  do if  $\text{docID}(p_1) = \text{docID}(p_2)$ 
4      then  $\text{ADD}(answer, \text{docID}(p_1))$ 
5           $p_1 \leftarrow \text{next}(p_1)$ 
6           $p_2 \leftarrow \text{next}(p_2)$ 
7  else if  $\text{docID}(p_1) < \text{docID}(p_2)$ 
8      then if  $\text{hasSkip}(p_1)$  and  $(\text{docID}(\text{skip}(p_1)) \leq \text{docID}(p_2))$ 
9          then while  $\text{hasSkip}(p_1)$  and  $(\text{docID}(\text{skip}(p_1)) \leq \text{docID}(p_2))$ 
10             do  $p_1 \leftarrow \text{skip}(p_1)$ 
11             else  $p_1 \leftarrow \text{next}(p_1)$ 
12 else if  $\text{hasSkip}(p_2)$  and  $(\text{docID}(\text{skip}(p_2)) \leq \text{docID}(p_1))$ 
13     then while  $\text{hasSkip}(p_2)$  and  $(\text{docID}(\text{skip}(p_2)) \leq \text{docID}(p_1))$ 
14         do  $p_2 \leftarrow \text{skip}(p_2)$ 
15         else  $p_2 \leftarrow \text{next}(p_2)$ 
16 return  $answer$ 
    
```


思考题

- 跳表指针是否适用于或查询 $x \text{ OR } y$?

跳表指针的位置

- 指针数目过多过少都不合适，要有一个均衡性：
 - 指针越多 → 跳步越短 ⇒ 更容易跳转，但是需要更多的与跳表指针指向记录的比较
 - 指针越少 → 比较次数越少，但是跳步越长 ⇒ 成功跳转的次数少



跳表指针的位置

- 简单的启发式策略：对于长度为 L 的倒排记录表，每 \sqrt{L} 处放一个跳表指针，即均匀放置。均匀放置方法忽略了查询词项的分布情况
- 如果索引相对静态，均匀方式方法是一种很简便的方法，但是如果索引经常更新造成 L 经常变化，均匀方式方式就很不方便
- 跳表方式在过去肯定是有用的，但是对于现代的硬件设备而言，如果合并的倒排记录表不能全部放入内存的话，上述方式不一定有用 (Bahle et al. 2002)
 - 更大的倒排记录表(含跳表)的 I/O开销可能远远超过内

提纲

① 上一讲回顾

② 文档

③ 词项

- 通常做法+非英语处理
- 英语

④ 跳表指针

⑤ 短语查询

短语查询及位置索引

PHRASE QUERIES AND POSITIONAL INDEXES

短语查询

- 输入查询作为一个短语整体，比如 “stanford university” “中国科学院”
- 因此，句子 “I went to university at Stanford” 就不应该是答案 （ “我去了中国农业科学院” ）
 - 有证据表明，用户很容易理解短语查询的概念，这也是很多搜索引擎”高级搜索”中比较成功的一个功能。
 - 但是很多查询是隐式短语查询， information retrieval textbook → [information retrieval] textbook
- 这种情况下，倒排记录表中仅保存docID是不够的
 - term + docIDs
- 有两种方法

第一种方法: 双词(Biword)索引

- 每两个连续的词组成词对(作为短语)来索引
- 比如文本片段 “Friends, Romans, Countrymen” 会产生两个词对
 - friends romans
 - romans countrymen
- 索引构建时, 将每个词对看成一个词项放到词典中
- 这样的话, 两个词组成的短语查询就能直接处理

更长的短语查询如何处理？

- 例子：查询请求stanford university palo alto,
 - 处理方法： 将其拆分成基于双词的布尔查询式：

stanford university AND

university palo AND palo alto

- 如果不对结果集做过滤，无法确认满足上述表达式的文档是否真正满足上述四个词组成的短语查询。也就是说满足上述布尔表达式只是满足短语查询的必要条件。

很难避免伪正例的出现！

扩展的双词 (Extended Biword)

- 对待索引文档进行词性标注
- 将词项进行组块，每个组块包含名词 (N) 和冠词/介词 (X)
- 称具有NX*N形式的词项序列为扩展双词(extended biword)
 - 将这样扩展词对作为词项放入词典中
- 例子: catcher in the rye (书名: 麦田守望者)
 - N X X N
- 查询处理: 将查询也分析成 N和X序列
 - 将查询切分成扩展双词
 - 在索引中查找: catcher rye

关于双词索引

- 会出现伪正例
- 由于词典中词项数目剧增，导致索引空间也激增
 - 如果3词索引，那么更是空间巨大，无法忍受
- 双词索引方法并不是一个标准的做法 (即倒排索引中一般不会全部采用双词索引方法)，但是可以和其他方法混合使用

第二种方法: 位置 (Positional)索引

- 在倒排记录表中，对每个词项在每篇文档中的每个位置(偏移或者单词序号)进行存储:
 - <词项, 出现词项的文档篇数;
 - doc1: 位置1, 位置2 ... ;
 - doc2: 位置1, 位置2 ... ;
 - ...>
 - 每个词项的倒排记录中的每一项，除了包括docID外，还包括一个位置列表

位置索引的例子

- 查询: “to₁ be₂ or₃ not₄ to₅ be₆”

<to, 993427:

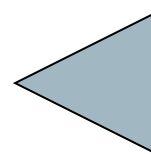
1: 7, 18, 33, 72, 86, 231;

2: 1, 17, 74, 222, 255;

4: 8, 16, 190, 429, 433;

5: 363, 367;

7: 13, 23, 191; ... >



1,2,4,5,7这几篇文章
中哪篇包含 “to be
or not to be”?

<be, 178239:

1: 17, 25;

4: 17, 191, 291, 430, 434;

5: 14, 19, 101; ... >

短语查询的处理

- 不仅仅简单按docID进行合并，还要考虑位置匹配
 - 找出同时包含这些词项的文档
 - 对每个词项，抽出其对应的倒排记录
 - 检查这些文档的位置列表，看各个词项之间是否存在与查询请求相同的位置关系
 - 例如，是否在某个be的前面一个词条位置上正好出现了to

短语查询的处理

- 查询: “to₁ be₂ or₃ not₄ to₅ be₆”

<to, 993427:

1: 7, 18, 33, 72, 86, 231;

2: 1, 17, 74, 222, 255;

4: 8, 16, 190, 429, 433;

5: 363, 367;

7: 13, 23, 191; ... >

<be, 178239:

1: 17, 25;

4: 17, 191, 291, 430, 434;

5: 14, 19, 101; ... >

求词项交集, 返回1,4,5



<to: ...; 4: ..., 429, 433; ... >

<be: ...; 4: ..., 430, 434; ... >



文章4满足要求

邻近搜索(Proximity query)

- 位置索引还可用于邻近搜索中，其搜索策略与短语查询类似，不同的是此时应考虑前后位置之间的距离不大于某个值
- 例如：employment /4 place
 - /k 表示“在 k 个词之内”
 - 包含两个词，且两个词之间距离小于四个词
- ***Employment*** agencies that ***place*** healthcare workers are seeing growth. (✓)
- **Employment** agencies that have learned to adapt now ***place*** healthcare workers is not a hit. (✗)

邻近搜索倒排记录表合并算法

```

POSITIONALINTERSECT( $p_1, p_2, k$ )
1   $answer \leftarrow \langle \rangle$ 
2  while  $p_1 \neq \text{NIL}$  and  $p_2 \neq \text{NIL}$ 
3  do if  $\text{docID}(p_1) = \text{docID}(p_2)$ 
4      then  $I \leftarrow \langle \rangle$ 
5           $pp_1 \leftarrow \text{positions}(p_1)$ 
6           $pp_2 \leftarrow \text{positions}(p_2)$ 
7          while  $pp_1 \neq \text{NIL}$ 
8          do while  $pp_2 \neq \text{NIL}$ 
9              do if  $|\text{pos}(pp_1) - \text{pos}(pp_2)| \leq k$ 
10                 then  $\text{ADD}(I, \text{pos}(pp_2))$ 
11                 else if  $\text{pos}(pp_2) > \text{pos}(pp_1)$ 
12                     then break
13                      $pp_2 \leftarrow \text{next}(pp_2)$ 
14                 while  $I \neq \langle \rangle$  and  $|I[0] - \text{pos}(pp_1)| > k$ 
15                     do  $\text{DELETE}(I[0])$ 
16                     for each  $ps \in I$ 
17                         do  $\text{ADD}(answer, \langle \text{docID}(p_1), \text{pos}(pp_1), ps \rangle)$ 
18                      $pp_1 \leftarrow \text{next}(pp_1)$ 
19                  $p_1 \leftarrow \text{next}(p_1)$ 
20                  $p_2 \leftarrow \text{next}(p_2)$ 
21             else if  $\text{docID}(p_1) < \text{docID}(p_2)$ 
22                 then  $p_1 \leftarrow \text{next}(p_1)$ 
23                 else  $p_2 \leftarrow \text{next}(p_2)$ 
24  return  $answer$ 
    
```


位置索引的大小

- 位置索引增加了位置信息，因此空间较大，但是可以采用索引压缩技术进行处理(参见第五讲)
- 当然，相对于没有位置信息的索引，位置索引的存储空间明显大于无位置信息的索引
- 另外，位置索引目前是实际检索系统的标配，这是因为实际中需要处理短语(显式和隐式)和邻近式查询

位置索引的大小

- 词项在每篇文档中的每次出现都需要一个存储单元
- 因此索引的大小依赖于文档的平均长度
 - 平均Web页面的长度 <1000 个词项
 - 美国证监会文件(SEC filings), 书籍, 甚至一些史诗 ... 和容易就超过 100,000 个词项
- 假设平均1000个词项中, 每个词项的频率都是1



文档大小	平均每个词项的倒排记录数	平均每个词项的位置索引存储单元
1000	1	1
100,000	1	100

一些经验规律

- 位置索引的大小大概是无位置信息索引的2-4倍
- 位置索引（压缩后）大概是原始文本容量的35-50%
- 提醒：上述经验规律适用于英语及类英语的语言

混合索引

- 上述两种索引方式可以混合使用
 - 对某些特定的短语 (如“Michael Jackson”, “Britney Spears”), 采用位置索引的方式效率不高
 - 还有“The Who” (英国一著名摇滚乐队), 采用位置索引, 效率更低
- Williams et al. (2004)对一种混合的索引机制进行了评估, 采用混合机制后
 - 相对于只使用位置索引, 空间开销额外增加了26%
 - 但对于典型的Web短语混合查询来说, 相对于只使用位置索引而言, 仅需要其 $\frac{1}{4}$ 的时间

本讲小结

- 理解典型信息检索系统中的两个基本单元：词项和文档
 - 什么是文档？什么是词项？
- 如何对索引文档进行处理来得到词典（词项）
 - 词条化(Tokenization)
 - 词条归一化（Normalization）
- 倒排记录表
 - 更快的合并算法：跳表法(skip list)
 - 短语查询的处理及带位置信息的倒排索引
 - 邻近搜索

参考资料

- 《信息检索导论》第2章
- MG 3.6, 4.3; MIR 7.2
- Porter's stemmer:
<http://www.tartarus.org/~martin/PorterStemmer/>
- 跳表理论: Pugh (1990)
 - Multilevel skip lists give same $O(\log n)$ efficiency as trees
- H.E. Williams, J. Zobel, and D. Bahle. 2004. "Fast Phrase Querying with Combined Indexes", ACM Transactions on Information Systems.
- <http://www.seg.rmit.edu.au/research/research.php?author=4>
- D. Bahle, H. Williams, and J. Zobel. Efficient phrase querying with an auxiliary index. SIGIR 2002, pp. 215-221.

课后练习

- 教材中习题 2-3、2-6、2-8、2-9
- 思考题：教材中习题2-14：
 - IR系统中如何同时使用位置索引和停用词表？潜在问题是什么？如何解决？

讨论议题

- 阅读作业1
 - 下一个十年的信息检索
 - Vannevar Bush "As we may think"
 - 信息检索的信念和偏见
 - "Beliefs and biases in web search", SIGIR' 2013 best paper
- 可以分组汇报，也可个人汇报