信息检索与Web搜索

第3讲 词项词典及倒排记录表

The term vocabulary and postings lists

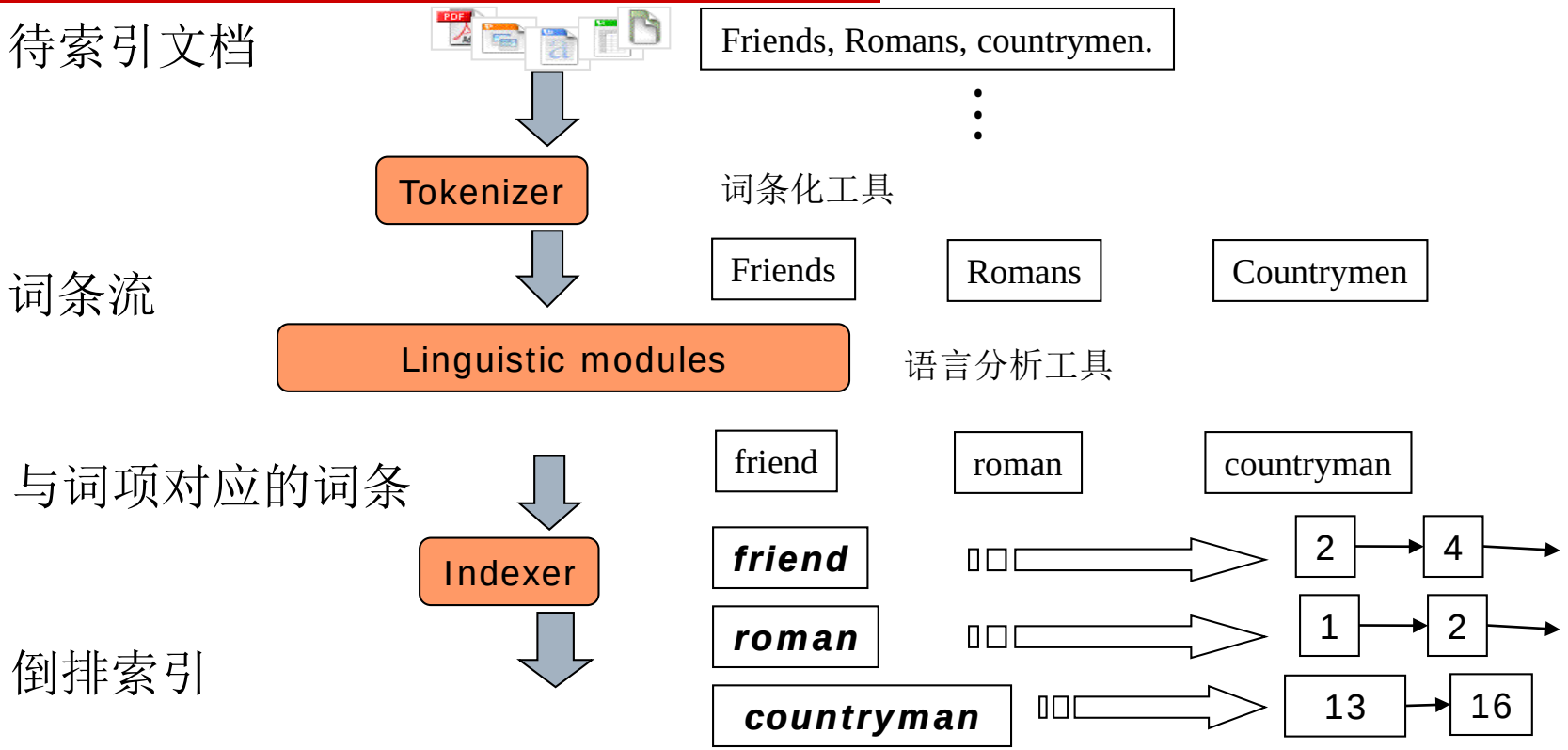
授课人：高曙明

*改编自“现代信息检索”网上公开课件（http://ir.ict.ac.cn/~wangbin）

主要内容

- 词项词典的有效构建
- 倒排记录表的扩展表示
- 对复杂查询的支持

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
与词项对应的词条
friend
roman
countryman
Indexer
friend
2
4
roman
1
2
countryman
13
16
倒排索引

文档预处理

- **任务目标：** 将文档转化成词项集合，支持倒排索引的构建，支持基于词项比对的文本检索
- **主要内容**
  - 文档编码转换
  - 文档单位选择
  - 文本分析：词条化、词条归一化、词形归并等
- **主要方法：** 自然语言处理、语言学

文档编码转换

- 任务：将字节序列转换成线性的字符序列
- 难点：多格式、多语言并存
- 方法：采用启发式方法进行文档语言识别、文档编码识别，再分类转换

文档单位选择

- 任务：确定被索引文档的合理粒度
- 粒度过大： 准确率低

  粒度过小：召回率低
- 方法：提供不同文档粒度，由用户根据实际需要选择合理的文档粒度

词条化(Tokenization)

- 任务：将字符序列分割成一系列有意义的子序列
- 例子：
  - 输入: “Friends, Romans and Countrymen”
  - 输出: **Friends Romans Countrymen**
- 词条：一个字符串实例，具有语义，适合作为索引单元
- 作用：词条化工作是构建倒排索引的基础，并影响检索效果
- 难点：如何有效地确定词条

词条化面对的问题

- “’”如何处理
  - O’Neill、boys’、Chile’s、aren’t
  - Finland’s capital → Finland? Finlands? Finland’s?
- “-”如何处理
  - Hewlett-Packard → 看成Hewlett 和 Packard 两个词条?
- 空格如何处理
  - San Francisco 到底是一个还是两个词条？
    - 如何判断是一个词条？
- 特殊词条如何识别
  - C++、C#、B-52、M*A*S*H、电话号码、网址… …

中文分词(Chinese Word Segmentation)

- 难点：没有空格符，字也可能有语义
- 例子：“和尚”、“和”、“尚”
- 方法：词典、统计学习、启发式规则、字词混合方式/k-gram（K字符序列）
- 例子：李明天天都准时上班
  - 李 明 天 天 都 准 时 上 班
  - 李明 天天 都 准时 上班
- 一般原则：没把握的情况下细粒度优先
- 一个策略：查询和文档采用一致的分词方法

停用词问题

- 停用词：在进行文档和查询匹配时作用不大的常见词
  - 一般不包含语义信息的词：the、a、and、to、be
  - 这些词都是高频词：前30个词就占了 ~30% 的倒排记录表空间
- 停用词去除的作用
  - 压缩索引空间
  - 提高检索响应速度
- 停用词确定方法
  - 高文档频率且与文档主题关系不大的词
  - 应用领域相关：“click”作为锚文本的停用词
  - 在处理查询时决定哪些词不用

停用词问题

- 现代信息检索系统中倾向于不去掉停用词
  - 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”
  - 在保留停用词的情况下，采用良好的压缩技术后，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，基本不会增加查询处理的开销

词条归一化(Normalization)

- 任务：将本质上等价但形式上不完全一致的多个词条归纳成一个等价类，即词项
- 作用：提高检索效果，缩小索引空间
- 两类方法：规则法、关联关系法
- 规则法：采用隐式规则在处理文档和查询时将多个词条映射成同一词项，比如：
  - 剔除句点规则：**U.S.A.，USA ⟶ USA**
  - 剔除连接符规则：
    anti-discriminatory, antidiscriminatory ⟶ antidiscriminatory

词条归一化(Normalization)

- 关联关系法：通过维护等价的非归一化词条之间的关联关系，显式地建立并应用等价类，如建立并应用同义词词典（Thesauri, WordNet）
  - 每一不重复的词条都作为索引单元
  - 处理查询时对每一词项基于等价类进行扩展，并将扩展后得到的多个词所对应的倒排表合到一起
  - 在索引构建时就对词进行扩展，如对于包含car的文档也放入automobile的倒排表中
- 方法比较：规则法效率高，关联关系法更灵活

归一化相关问题及其处理

- **重音符问题：** 如法语中 **résumé vs. resume**
  - 处理方法：常常归一化成不带重音符号的形式
    - **Tuebingen, Tübingen, Tubingen --→ Tubingen**
- **大小写问题：** **Automobile vs. automobile**
  - 处理方法：将句首词转换成小写形式，将标题中大写或首字母大写的全部单词转换成小写形式
- **时间格式问题：** **3/12/91, Mar. 12, 1991**（美式），**12/3/91**（欧式）
  - 处理方法：启发式规则+语言识别

词形归并(Lemmatization)

- 任务：将单词的不同语法形态还原为原形
- 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**
- 作用：提高检索效果，减少索引单元
- 方法：词典+相关词列表
- 问题：如：**found → find? found?**
  - 上下文语义理解?

词干还原（Stemming）

- 任务：将词项归约(reduce)成其词干(stem)
  - 比如，将 **automate(s), automatic, automation**都还原成 **automat**
- 作用：提高检索效果，减少索引单元（**5-10% for English，50% in Arabie，30%**芬兰语）
- 方法：基于规则截除词缀

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，开始于70年代
- 一些规则+ 5 步骤的归约过程

Step 1a:

- Replace ***sses*** by ***ss*** (e.g., stresses → stress).
- Delete ***s*** if the preceding word part contains a vowel not immediately before the ***s*** (e.g., gaps → gap but gas → gas).
- Replace ***ied*** or ***ies*** by ***i*** if preceded by more than one letter, otherwise by ***ie*** (e.g., ties → tie, cries → cri).
- If suffix is ***us*** or ***ss*** do nothing (e.g., stress → stress).

Step 1b:

- Replace ***eed***, ***eedly*** by ***ee*** if it is in the part of the word after the first non-vowel following a vowel (e.g., agreed → agree, feed → feed).
- Delete ***ed***, ***edly***, ***ing***, ***ingly*** if the preceding word part contains a vowel, and then if the word ends in ***at***, ***bl***, or ***iz*** add ***e*** (e.g., fished → fish, pirating → pirate), or if the word ends with a double letter that is not ***ll***, ***ss*** or ***zz***, remove the last letter (e.g., falling→ fall, dripping → drip), or if the word is short, add ***e*** (e.g., hoping → hope).
- Whew!

Porter算法

- **Porter算法问题举例**

False positives	*False negatives*
organization/organ	european/europe
generalization/generic	cylinder/cylindrical
numerical/numerous	matrices/matrix
policy/police	urgency/urgent
university/universe	create/creation
addition/additive	analysis/analyses
negligible/negligent	useful/usefully
execute/executive	noise/noisy
past/paste	decompose/decomposition
ignore/ignorant	sparse/sparsity
special/specialized	resolve/resolution
head/heading	triangle/triangular

[Note] *false positive: pairs of different words have the same stem; false negative: pairs of words have different stem when they should have the same.*

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/Husk**

  http://www.cs.waikato.ac.nz/~eibe/stemmers
- 基于词形分析对于检索只能提供一定的帮助
- 未来：基于词用分析？

词条化及其归一化：实例

- 文本：

  We love you because of what you do for us

- 词条：

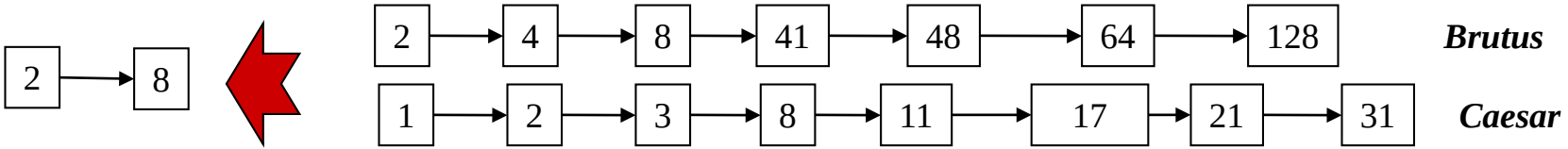
  We love you because of what you do for us

- 词项：

  We love you because do

回顾：基本合并算法

- 两个指针，同步扫描，线性时间

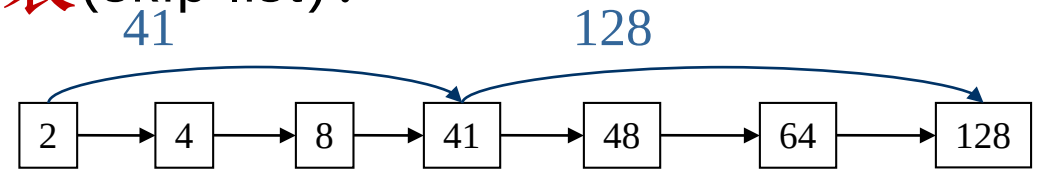


两个表长度为m和n的话，上述合并时间复杂度为 $O(m+n)$

能否做得更好？

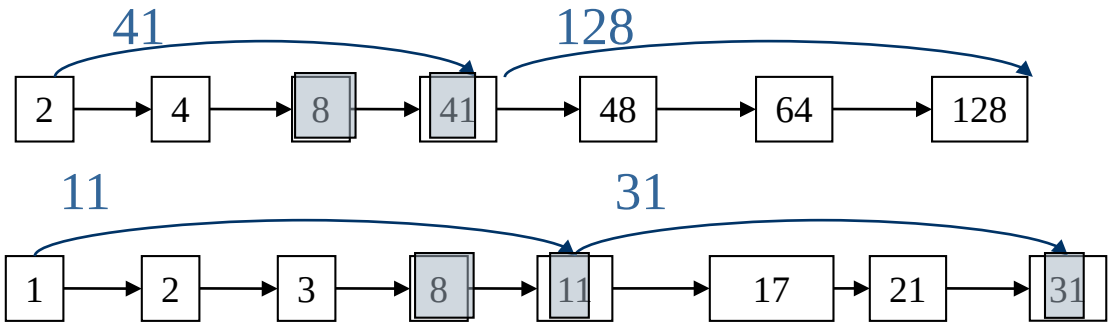
带跳表指针的倒排表

- 跳表(skip list)：



- 作用：支持在遍历倒排表时跳过那些不可能出现在检索结果中的记录项，以提高合并操作的效率
- 需要解决的问题：如何利用跳表指针支持倒排表的快速合并？在什么地方加跳表指针？

基于跳表的倒排表快速合并



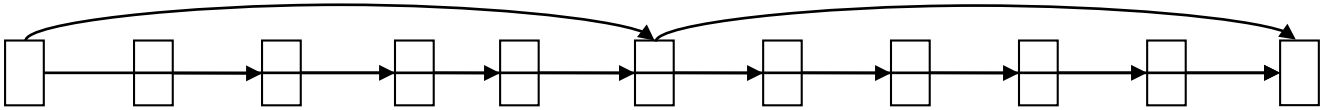
- 假定匹配到上下的指针都指向8，接下来两个指针都向下移动一位
- 比较41和11，这里11小且有跳表指针（指向31），则直接比较41和31，由于31仍然比41小，于是下指针直接指向31，这样就跳过了17、21两项

基于跳表的倒排表快速合并

```
INTERSECTWITHSKIPS(p1, p2)
 1  answer ← ⟨ ⟩
 2  while p1 ≠ NIL and p2 ≠ NIL
 3  do if docID(p1) = docID(p2)
 4        then ADD(answer, docID(p1))
 5             p1 ← next(p1)
 6             p2 ← next(p2)
 7        else if docID(p1) < docID(p2)
 8                then if hasSkip(p1) and (docID(skip(p1)) ≤ docID(p2))
 9                        then while hasSkip(p1) and (docID(skip(p1)) ≤ docID(p2))
10                                do p1 ← skip(p1)
11                        else p1 ← next(p1)
12                else if hasSkip(p2) and (docID(skip(p2)) ≤ docID(p1))
13                        then while hasSkip(p2) and (docID(skip(p2)) ≤ docID(p1))
14                                do p2 ← skip(p2)
15                        else p2 ← next(p2)
16  return answer
```

跳表指针的位置选择

- **均衡性：**指针数目过多过少都不合适
  - 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
- **简单的启发式策略：**对于长度为L的倒排记录表，每 √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



短语查询

- 短语查询：以一个短语整体作为查询的查询方式，比如“stanford university” “浙江大学”
- 是一种符合人们需要的查询方式
- 基于关键词的查询难以达到短语查询的效果
- 原因何在？
- 问题：如何改造索引？如何识别文档中的短语？

双词(Biword)索引

- 目的：将文档中每两个连续的词组成词对，作为索引单元
- 例子：对文本片段“Friends, Romans, Countrymen”，产生两个词对
  - **friends romans**
  - **romans countrymen**
- 方法：索引构建时，将每个词对看成一个词项放到词典中
- 问题：大大增加词汇表的规模和索引的规模

更长的短语查询处理

- 方法：将其拆分成基于双词的布尔查询，拆分采用K-gram策略
- 例子： 对于stanford university palo alto，将其拆分成 stanford university AND university palo AND palo alto 进行查询处理
- 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！

双词扩展（Extended Biword）

- 目的：有效支持名词短语查询
- 扩展方法：
  - 对文档进行词性标注，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，放入词典
- 例子：catcher in the rye

  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

位置索引(Positional indexes)

- 带词项位置信息的倒排索引，即在倒排记录表中，对每个词项在每篇文档中的每个位置(单词序号)进行存储

<term, 出现term的文档篇数;
doc1: 位置1, 位置2 … ;
doc2: 位置1, 位置2 … ;
等等>

<*be*: 993427;
1: 7, 18, 33, 72, 86, 231;
2: 3, 149;
4: 17, 191, 291, 430, 434;
5: 363, 367, …>

- 目的：一般性地支持短语查询和近邻查询

基于位置索引的短语查询处理

- 短语查询：“to be or not to be”
- 对每个词项，抽出其对应的倒排记录表: to, be, or, not
- 合并倒排记录表，考虑位置匹配（保持位置关系一致）
  - to:
    - 2:1,17,74,222,551; 4:8,16,190,429,433; 7:13,23,191; ...
  - be:
    - 1:17,19; 4:17,191,291,430,434; 5:14,19,101; ...
- K近邻搜索中的搜索策略与此类似，不同的是此时考虑前后位置之间的距离不大于K

基于位置索引的合并算法

```
POSITIONALINTERSECT(p1, p2, k)
 1  answer ← ⟨ ⟩
 2  while p1 ≠ NIL and p2 ≠ NIL
 3  do if docID(p1) = docID(p2)
 4        then l ← ⟨ ⟩
 5             pp1 ← positions(p1)
 6             pp2 ← positions(p2)
 7             while pp1 ≠ NIL
 8             do while pp2 ≠ NIL
 9                do if |pos(pp1) − pos(pp2)| ≤ k
10                      then ADD(l, pos(pp2))
11                      else if pos(pp2) > pos(pp1)
12                              then break
13                   pp2 ← next(pp2)
14                while l ≠ ⟨ ⟩ and |l[0] − pos(pp1)| > k
15                do DELETE(l[0])
16                for each ps ∈ l
17                do ADD(answer, ⟨docID(p1), pos(pp1), ps⟩)
18                pp1 ← next(pp1)
19             p1 ← next(p1)
20             p2 ← next(p2)
21        else if docID(p1) < docID(p2)
22                then p1 ← next(p1)
23                else p2 ← next(p2)
24  return answer
```

位置索引分析

- 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和邻近式查询
- 位置索引需要更大的存储空间，因为增加了位置信息，但是可以采用索引压缩技术进行处理
- 位置索引的大小大概是无位置信息索引的2-4倍
- 位置索引大概是原始文本容量的35-50%
- 提高了倒排记录表合并操作的复杂性

混合索引

- **混合索引**：将双词索引和位置索引合并形成的索引，其中双词为用户查询中的高频双词
- **目的**：提高检索效率
  - 对某些特定的短语 (如“Michael Jackson”, “Britney Spears”)，如果采用位置索引的方式那么效率不高
  - 对于“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)的评估结果**
  - 采用混合机制，那么对于典型的Web查询(比例)来说，相对于只使用位置索引而言，仅需要其¼ 的时间
  - 相对于只使用位置索引，空间开销只增加了26％

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.

课后作业

- 见课程网页:

  http://10.76.3.31