信息检索导论

An Introduction to Information Retrieval

第3讲 词典及容错式检索

Dictionary and tolerant retrieval

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

*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

- 上一讲回顾
- 词典
- 通配查询
- 编辑距离
- 拼写校正
- Soundex

提纲

- 上一讲回顾
- 词典
- 通配查询
- 编辑距离
- 拼写校正
- Soundex

上一讲内容

- 文档
- 词条/词项
- 基于跳表指针的合并
- 短语查询的处理(双词索引和位置索引)

文档

- 索引的基本单位
  - 与文件不是一回事，严格地说，一篇文档可能包含多个文件，也可能一个文件包含多篇文档
  - 依赖于具体应用
    - 句子级检索： 一个句子为一篇文档
    - 段落级检索： 一段文本为一篇文档
    - ……

词类(type)/词条(token)的区别

- 词条(Token) – 词或者词项在文档中出现的实例，出现多次算多个词条
- 词类(Type) – 多个词条构成的等价类(equivalence class)集合
- *In June, the dog likes to chase the cat in the barn.*
- 12 个词条, 9个词类

- 词类经过一些处理(去除停用词、归一化)之后，最后用于索引的称为为词项

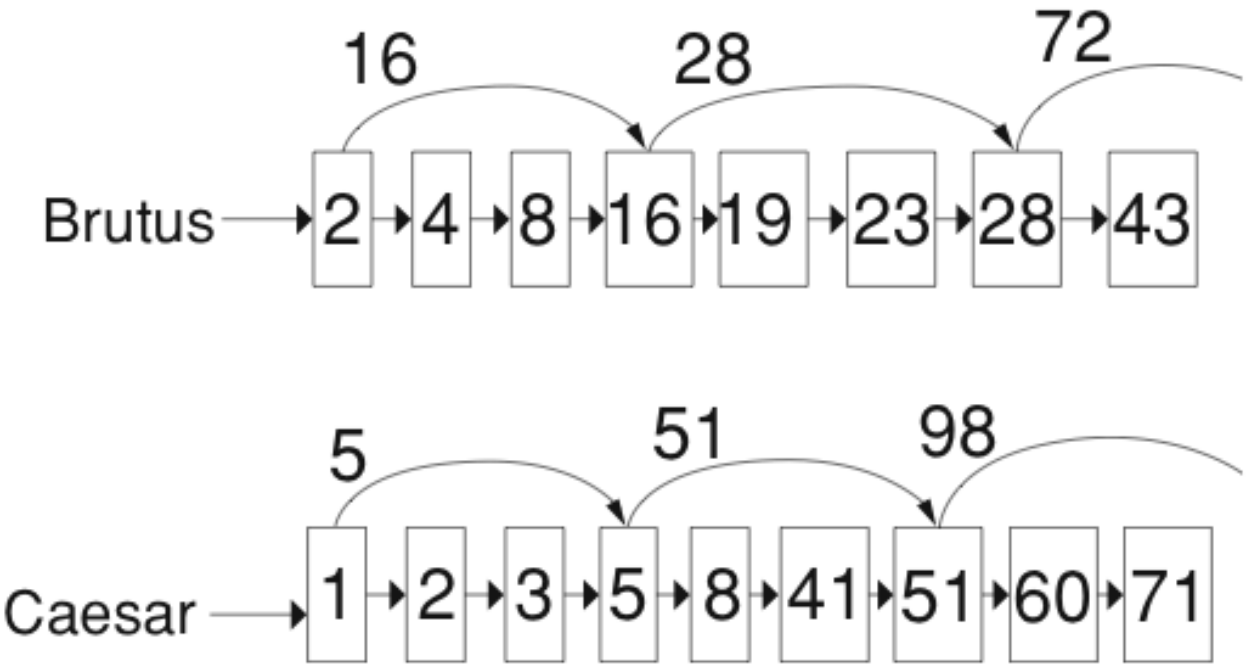
词条化中考虑的问题

- 词之间的边界是什么？空格？撇号还是连接符？
- 上述边界不一定是真正的边界 (比如, 中文)
- 另外荷兰语、德语、瑞典语复合词中间没有空格 (*Lebensversicherungsgesellschaftsangestellter*)

词项归一化中的问题

- 词项实际上是一系列词条组成的等价类
- 如何定义等价类？
  - 数字 (3/20/91 vs. 20/3/91)
  - 大小写问题
  - 词干还原，Porter工具
  - 形态分析（词形归并）: 屈折变体 vs. 派生变体
- 其他语言中词项归一化的问题
  - 比英语中形态更复杂
  - 芬兰语: 单个动词可能有12,000 个不同的形式different forms
  - 重音符号、元音变音问题（umlauts，由于一个音被另一个音词化而导致的变化，尤其是元音的变化）

跳表指针



位置(信息)索引

- 在无位置信息索引中，每条倒排记录只是一个docID
- 在位置信息索引中，每条倒排记录是一个docID加上一个位置信息表
- 一个查询的例子: "$to_1\ be_2\ or_3\ not_4\ to_5\ be_6$"

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

位置信息索引

- 基于位置信息索引，能够处理
  - 短语查询(phrase query)
  - 邻近查询(proximity query)

```
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
```

l：p2中，与pp1距离小于k的位置集合

遍历倒排表，对于词项交集中的文档

对词项1对应的每个文档位置，遍历词项2的文档位置

词项间隔<=k

加入结果集

位置信息索引

TO, 993427:
‹ 4: ‹8, 16, 18, 190, 429, 433›;
. . . ›
BE, 178239:
‹ 4: ‹17, 20, 191, 291, 430, 434›;
. . . ›
Q: TO /2 BE

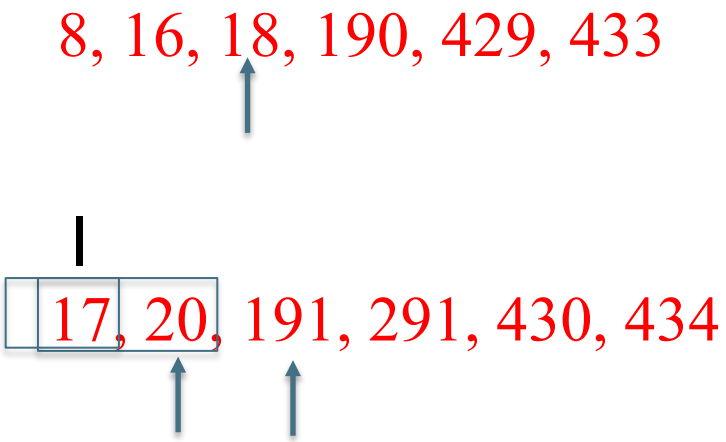
8, 16, 18, 190, 429, 433

输出：
<4, 16, 17>

17, 20, 191, 291, 430, 434

位置信息索引

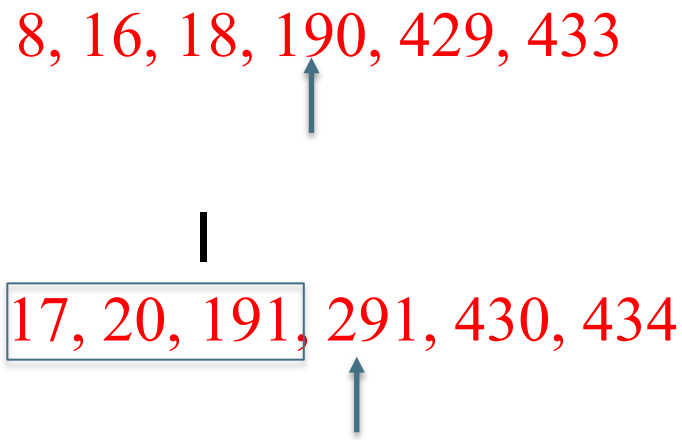
TO, 993427:
‹ 4: ‹8, 16, 18, 190, 429, 433›;
. . . ›
BE, 178239:
‹ 4: ‹17, 20, 191, 291, 430, 434›;
. . . ›
Q: TO /2 BE



输出：
<4, 16, 17>
<4, 18, 17>
<4, 18, 20>

位置信息索引

TO, 993427:
‹ 4: ‹8, 16, 18, 190, 429, 433›;
. . . ›
BE, 178239:
‹ 4: ‹17, 20, 191, 291, 430, 434›;
. . . ›
Q: TO /2 BE



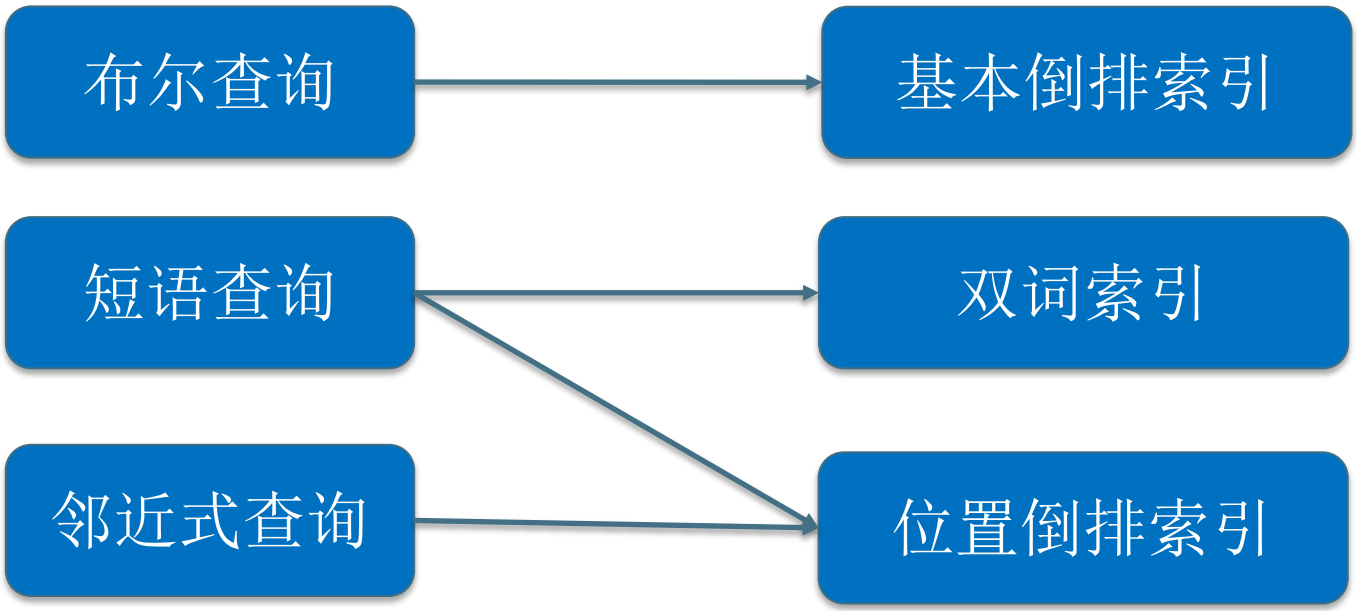
输出：
<4, 16, 17>
<4, 18, 17>
<4, 18, 20>
<4, 190,191>
……

位置信息索引

- 假设：
  - P为倒排记录表长度之和
  - 两个倒排记录表的长度分别x、y
- 临近搜索算法的时间复杂度是多少？
- 如果仅输出命中docID，时间复杂度是多少？

```
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
```

查询 vs. 索引(迄今为止)



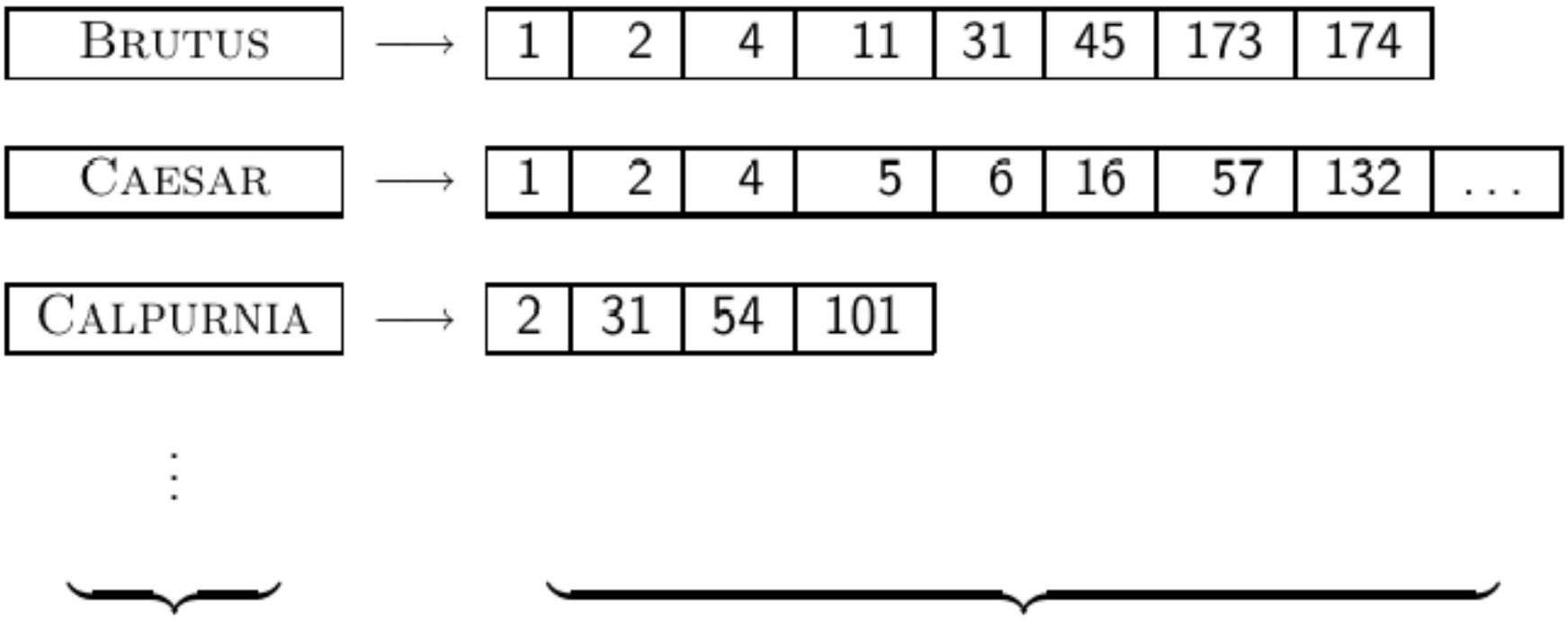
本讲内容

- 词典的数据结构：访问效率和支持查找的方式
- 容错式检索(Tolerant retrieval): 如果查询词项和文档词项不能精确匹配时如何处理？
  - 通配查询：包含通配符*的查询
  - 拼写校正：查询中存在错误时的处理

提纲

- 上一讲回顾
- 词典
- 通配查询
- 编辑距离
- 拼写校正
- Soundex

倒排索引


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(posting list)

词典

- 词典是指存储词项词汇表的数据结构
- 词项词汇表(Term vocabulary): 指的是具体数据（词项）
- 词典(Dictionary): 指的是数据结构

采用定长数组的词典结构

- 对每个词项，需要存储:
  - 文档频率
  - 指向倒排记录表的指针
  - . . .
- 暂定每条词项的上述信息均采用定长的方式存储

- 假定所有词项的信息采用数组存储

采用定长数组的词典结构

词项	文档频率	指向倒排记录表的指针
a	656 265	→
aachen	65	→
…	…	…
zulu	221	→

空间消耗： 20字节 4字节 4字节

词项定位(即查词典)

- 输入“信息”，如何在词典中快速找到这个词？
- 很多词典应用中的基本问题。
- 以下介绍支持快速查找的词典数据结构。

用于词项定位的数据结构

- 主要有两种数据结构：哈希表和树
- 有些IR系统用哈希表，有些系统用树结构
- 采用哈希表或树的准则：
  - 词项数目是否固定或者说词项数目是否持续增长？
  - 词项的相对访问频率如何？
  - 词项的数目有多少？

哈希表(散列表)

哈希函数，输入词项，输出正整数(通常是地址)

f(信息)=18, f(数据)=19,
f(挖掘)=19

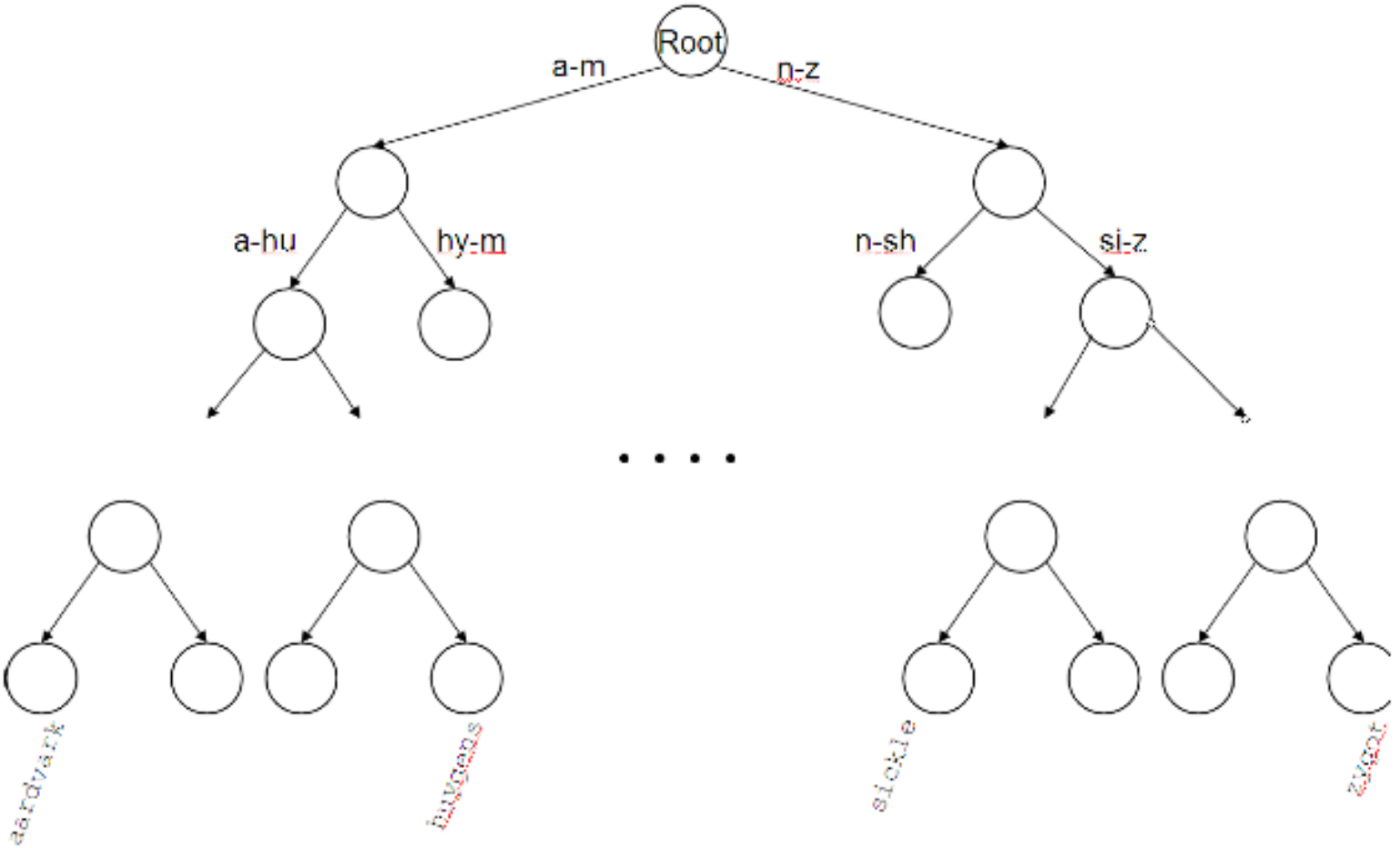
哈希表

- 每个词项通过哈希函数映射成一个整数
- 尽可能避免冲突
- 查询处理时：对查询词项进行哈希，如果有冲突，则解决冲突，最后在定长数组中定位
- 优点: 在哈希表中的定位速度快于树中的定位速度
  - 查询时间是常数
- 缺点：
  - 无法处理词项的微小变形 (*resume* vs. *résumé*)
  - 不支持前缀搜索 (比如所有以*automat*开头的词项)
  - 如果词汇表不断增大，需要定期对所有词项重新哈希

关于哈希

- 完美哈希→最小完美哈希→保序最小完美哈希
- 局部敏感哈希(locality sensitive hashing，LSH): 如SimHash
- 哈希学习(Hash Learning)：学习出哈希编码
- 哈希函数的用途
  - 查重(包括完全的重复和近似的重复)
  - 加密
  - 签名

树(Tree)



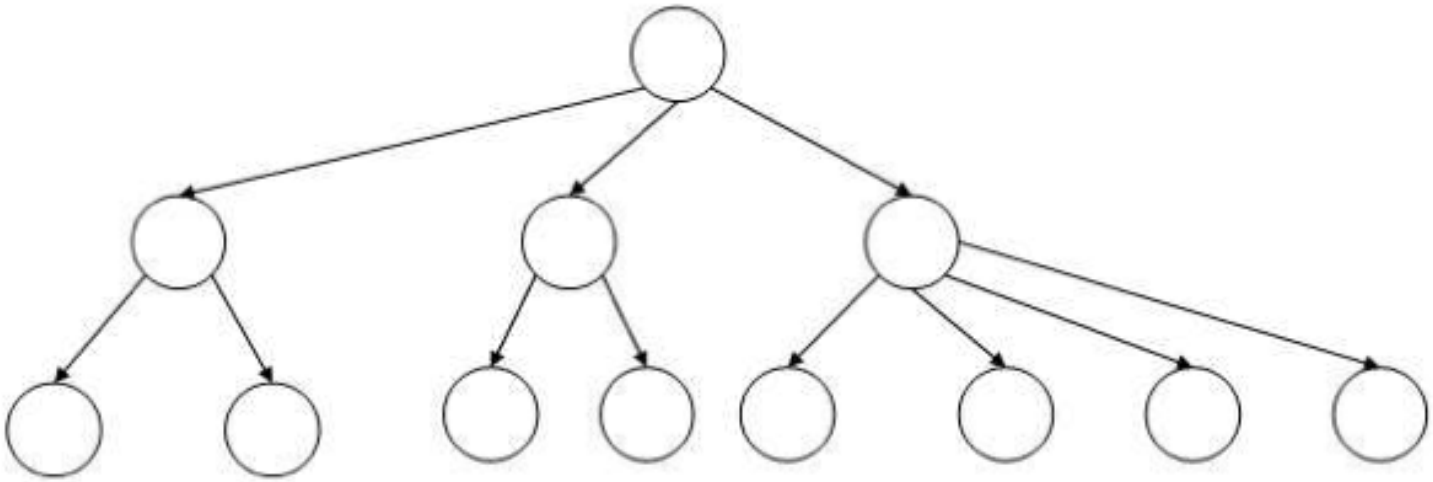
一棵二叉树(Binary Tree)

树结构的性质

- 优点：
  - 树可以支持前缀查找(相当于对词典再建一层索引)
- 缺点：
  - 二叉树的搜索速度略低于哈希表方式： $O(\log M)$, 其中 M 是词汇表大小，即所有词项的数目
  - 当然，$O(\log M)$ 仅仅对平衡树成立，使二叉树重新保持平衡开销很大

- B-树(Balanced Tree)能够缓解上述二叉树的问题
- B-树定义：每个内部节点的子节点数目在 $[a, b]$之间，其中 a, b 为合适的正整数, e.g., [2, 4].

B-树



- B-树的搜索速度为$O(\log M)$
- 但是没有平衡问题

提纲

- 上一讲回顾
- 词典
- 通配查询
- 编辑距离
- 拼写校正
- Soundex

通配查询的处理

- mon*: 找出所有包含以 *mon*开头的词项的文档
- 如果采用B-树词典结构，那么实现起来非常容易，只需要返回区间mon ≤ t < moo上的词项t
- *mon: 找出所有包含以*mon*结尾的词项的文档
  - 将所有的词项倒转过来，然后基于它们建一棵附加的树
  - 返回区间nom ≤ t < non上的词项t
- 也就说，通过上述数据结构，可能得到满足通配查询的一系列词项，然后返回任一词项的文档

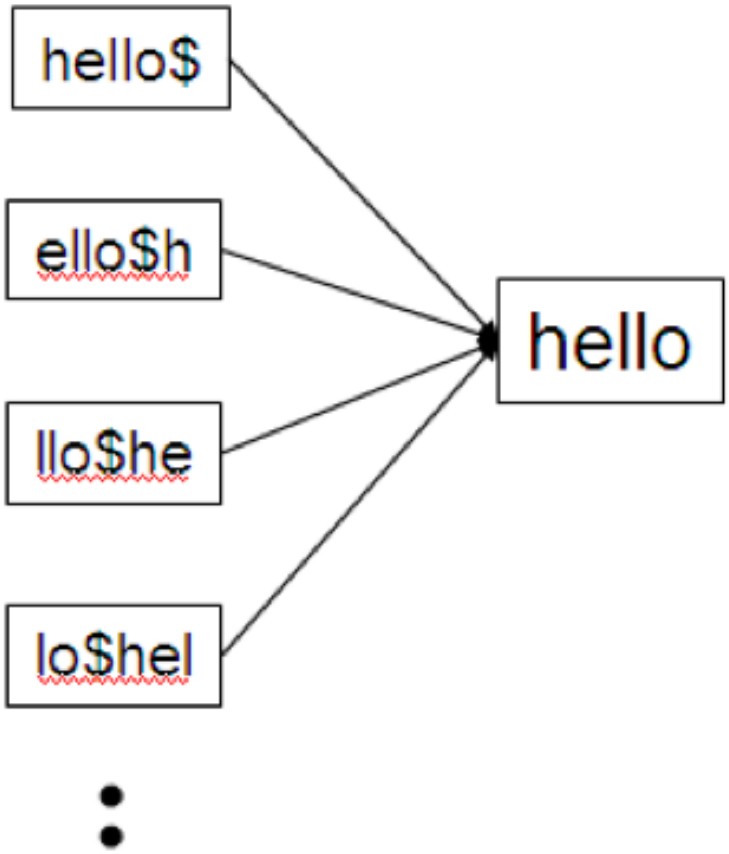
词项中间的 *号处理

- 例子: m*nchen
  - 在B-树中分别查找满足m*和 *nchen的词项集合，然后求交集
  - 这种做法开销很大
- 另外一种方法: 轮排(permuterm) 索引
- 基本思想：
  - 将通配查询词项旋转，使*出现在末尾
  - 将每个旋转后的结果存放在词典中，即B-树中

轮排索引

- 对于词项hello: 将 *hello$, ello$h, llohe, lohel, o$hell* 和 *$hello*加入到 B-树中，其中 $ 是一个特殊符号(表示结尾)
- 即在词项前面再加一层索引
  - 该索引采用B-树来组织
  - 该索引叶节点是词项的各种变形

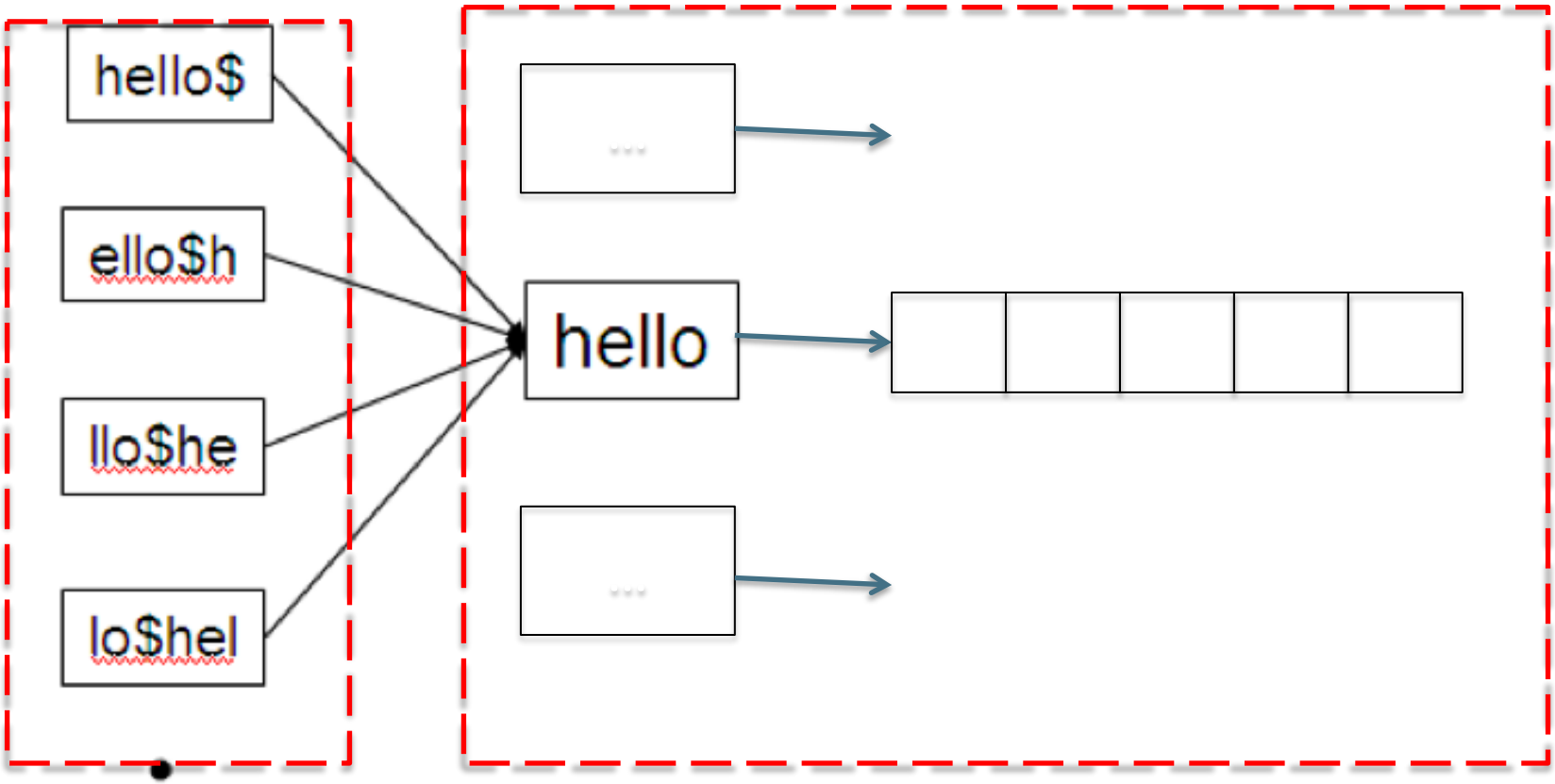
轮排结果 → 词项的映射示意图



轮排索引

- 对于hello，轮排索引中已经存储了 *hello$, ello$h, llohe, lohel, o$hell*和*$hello*字符串
- 查询处理
  - 输入查询为 X，则在轮排索引中寻找 X$字符串即可
  - 输入查询为 X*，则寻找以$X开始的字符串
  - 输入查询为 *X，则寻找以X$开始的字符串
  - 输入查询为 *X*，则寻找X开始的字符串即可，比如查询为*ello*，则只需要查到ello开头的串即可(上面是ello$h)，因为在轮排索引中，ello右部一定包含一个$，不论$是否处于尾部，该串均能满足查询*X*
  - 输入查询为 X*Y, 则寻找Y$X开始的字符串，比如通配查询为hel*o, 那么相当于要寻找o$hel开始的字符串
- 轮排索引称为轮排树更恰当，但是轮排索引的称呼已经使用非常普遍

轮排索引小结



轮排索引 (通配查询→词项，采用B树来组织)

传统倒排索引 (词项→文档)

使用轮排索引的查找过程

- 将查询进行旋转，将通配符旋转到右部
- 同以往一样查找B-树，得到匹配的所有词项，将这些词项对应的倒排记录表取出
- 问题：相对于通常的B-树，轮排索引(轮排树)的空间要大4倍以上 (经验值)

k-gram 索引

- 比轮排索引空间开销要小
- 枚举一个词项中所有连读的*k*个字符构成***k*-gram** 。
  - 2-gram称为二元组(**bigram**)
  - 3-gram称为三元组(**trigram**)
- 例子: *April is the cruelest month*
  - 2-gram: *$a ap pr ri il l$ $i is s$ $t th he e$ $c cr ru ue el le es st t$ $m mo on nt h$*
  - 同前面一样，$ 是一个特殊字符，表示单词开始或结束

k-gram索引

- 构建一个倒排索引，此时词典部分是所有的k-gram，倒排记录表部分是包含某个k-gram的所有词项
- 相当于对词项再构建一个倒排索引(二级索引)

k-gram (bigram, trigram, . . .) 索引

- 需要注意的是，这里有两个倒排索引
- 词典-文档的倒排索引基于词项返回文档
- 而*k*-gram索引用于查找词项，即基于查询所包含的*k*-gram来查找所有的词项

利用2-gram索引处理通配符查询

- 例子：查询mon*
  - 先执行布尔查询: $m AND mo AND on
  - 该布尔查询会返回所有以前缀*mon*开始的词项 . . .
  - . . . 当然也可能返回许多伪正例(false positives)，比如MOON。同前面的双词索引处理短语查询一样，满足布尔查询只是满足原始查询的必要条件。因此，必须要做后续的过滤处理
  - 剩下的词项将在词项-文档倒排索引中查找文档
- *k*-gram索引 vs. 轮排索引
  - *k*-gram索引的空间消耗小
  - 轮排索引不需要进行后过滤

课堂练习

- Google对通配符查询的支持极其有限
- 比如：在 Google中查询 [gen* universit*]
  - 意图：想查 University of Geneva, 但是不知道如何拼写，特别是法语中的拼写
- 按照Google自己的说法, 2010-04-29: “* 操作符只能作为一个整体单词使用，而不能作为单词的一部分使用”
- 但是这点并不完全对，尝试一下 [pythag*] 和 [m*nchen]
- 问题: 为什么Google对通配查询并不充分支持？

原因

- 问题 1: 一条通配符查询往往相当于执行非常多的布尔查询
  - 对于 [gen* universit*]: geneva university OR geneva université OR genève university OR genève université OR general universities OR . . .
  - 开销非常大
- 问题 2: 用户不愿意敲击更多的键盘
  - 如果允许[pyth* theo*]代替 [pythagoras' theorem]的话，用户会倾向于使用前者
  - 这样会大大加重搜索引擎的负担
  - Google Suggest是一种减轻用户输入负担的好方法

提纲

- 上一讲回顾
- 词典
- 通配查询
- 编辑距离
- 拼写校正
- Soundex

拼写错误的查询

- 用户想查information，但是错误地输入了informaton
- 于是，我们试图将informaton → information
- 需要使用拼写校正技术

拼写校正

- 两个主要用途
  - 纠正待索引文档
  - 纠正用户的查询
- 两种拼写校正的方法
- 词独立(**Isolated word**)法
  - 只检查每个单词本身的拼写错误
  - 如果某个单词拼写错误后变成另外一个单词，则无法查出，e.g., *an asteroid that fell **form** the sky*
- 上下文敏感(**Context-sensitive**)法
  - 纠错时要考虑周围的单词
  - 能纠正上例中的错误 *form/from*

关于文档校正

- 本课当中我们不关心文档的拼写校正问题 (e.g., MS Word)
- 在IR领域, 我们主要对OCR处理后的文档进行拼写校正处理. (OCR = optical character recognition，光学字符识别)
- IR领域的一般做法是：不改变文档

查询校正

- 第一种方法: 词独立(isolated word)法
  - 假设1: 对需要纠错的词存在一系列“正确单词形式”
  - 假设2: 需要提供存在错误拼写的单词和正确单词之间的距离计算方式

- 简单的拼写校正算法: 返回与错误单词具有最小距离的”正确”单词
  - 例子: i*nformaton* → *information*
  - 可以将词汇表中所有的单词都作为候选的“正确”单词

几种可用的词汇表

- 采用标准词典 (韦伯词典, 牛津词典等等)
- 采用领域词典 (面向特定领域的IR系统)
- 采用文档集上的词项词汇表(可以通过统计得到)，但是每个词项均带有权重

单词间距离的计算

- 以下将介绍几种计算方法
  - 编辑距离(Edit distance或者Levenshtein distance)
  - 带权重的编辑距离
  - *k*-gram 重叠率

编辑距离

- 两个字符串s_1和s_2编辑距离是指从 s_1 转换成s_2所需要的最少的基本操作数目
- Levenshtein距离: 采用的基本操作是插入(insert)、删除(delete)和替换(replace)
  - Levenshtein距离 *dog-do*: 1
  - Levenshtein距离 *cat-cart*: 1
  - Levenshtein距离 *cat-cut*: 1
  - Levenshtein距离 *cat-act*: 2
- Damerau-Levenshtein距离：除了上述三种基本操作外，还包括两个字符之间的交换（transposition）操作
  - Damerau-Levenshtein距离 *cat-act*: 1

Vladimir Iosifovich Levenshtein

- 俄罗斯科学家(1935-)
- 研究信息论、纠错理论
- 毕业于莫斯科国立大学
- 1965年提出Levenshtein距离
- 2006年获得IEEE Richard W. Hamming Medal

Levenshtein距离: 算法

LEVENSHTEINDISTANCE(s_1, s_2)

```
 1  for i ← 0 to |s1|
 2  do m[i, 0] = i
 3  for j ← 0 to |s2|
 4  do m[0, j] = j
 5  for i ← 1 to |s1|
 6  do for j ← 1 to |s2|
 7       do if s1[i] = s2[j]
 8            then m[i, j] = min{m[i-1, j]+1, m[i, j-1]+1, m[i-1, j-1]}
 9            else  m[i, j] = min{m[i-1, j]+1, m[i, j-1]+1, m[i-1, j-1]+1}
10  return m[|s1|, |s2|]
```

Operations: insert (cost 1), delete (cost 1), replace (cost 1), copy (cost 0)

(i-1,j-1)	(i-1,j)
(i,j-1)	(i,j)

Levenshtein距离: 算法

LEVENSHTEINDISTANCE(s_1, s_2)

```
 1  for i ← 0 to |s1|
 2  do m[i, 0] = i
 3  for j ← 0 to |s2|
 4  do m[0, j] = j
 5  for i ← 1 to |s1|
 6  do for j ← 1 to |s2|
 7       do if s1[i] = s2[j]
 8            then m[i, j] = min{m[i-1, j]+1, m[i, j-1]+1, m[i-1, j-1]}
 9            else  m[i, j] = min{m[i-1, j]+1, m[i, j-1]+1, m[i-1, j-1]+1}
10  return m[|s1|, |s2|]
```

Operations: insert (cost 1), delete (cost 1), replace (cost 1), copy (cost 0)

(i-1,j-1)	(i-1,j)
(i,j-1)	(i,j)

左邻居

Levenshtein距离: 算法

LEVENSHTEINDISTANCE(s_1, s_2)

```
 1  for i ← 0 to |s1|
 2  do m[i, 0] = i
 3  for j ← 0 to |s2|
 4  do m[0, j] = j
 5  for i ← 1 to |s1|
 6  do for j ← 1 to |s2|
 7       do if s1[i] = s2[j]
 8            then m[i, j] = min{m[i-1, j]+1, m[i, j-1]+1, m[i-1, j-1]}
 9            else  m[i, j] = min{m[i-1, j]+1, m[i, j-1]+1, m[i-1, j-1]+1}
10  return m[|s1|, |s2|]
```

Operations: insert (cost 1), delete (cost 1), replace (cost 1), copy (cost 0)

(i-1,j-1)	(i-1,j)
(i,j-1)	(i,j)

上邻居

Levenshtein距离: 算法

LEVENSHTEINDISTANCE(s_1, s_2)

```
 1  for i ← 0 to |s1|
 2  do m[i, 0] = i
 3  for j ← 0 to |s2|
 4  do m[0, j] = j
 5  for i ← 1 to |s1|
 6  do for j ← 1 to |s2|
 7       do if s1[i] = s2[j]
 8            then m[i, j] = min{m[i-1, j]+1, m[i, j-1]+1, m[i-1, j-1]}
 9            else  m[i, j] = min{m[i-1, j]+1, m[i, j-1]+1, m[i-1, j-1]+1}
10  return m[|s1|, |s2|]
```

Operations: insert (cost 1), delete (cost 1), replace (cost 1), copy (cost 0)

(i-1,j-1)	(i-1,j)
(i,j-1)	(i,j)

左上邻居

Levenshtein距离: 算法

LEVENSHTEINDISTANCE(s_1, s_2)

```
 1  for i ← 0 to |s1|
 2  do m[i, 0] = i
 3  for j ← 0 to |s2|
 4  do m[0, j] = j
 5  for i ← 1 to |s1|
 6  do for j ← 1 to |s2|
 7       do if s1[i] = s2[j]
 8            then m[i, j] = min{m[i-1, j]+1, m[i, j-1]+1, m[i-1, j-1]}
 9            else  m[i, j] = min{m[i-1, j]+1, m[i, j-1]+1, m[i-1, j-1]+1}
10  return m[|s1|, |s2|]
```

Operations: insert (cost 1), delete (cost 1), replace (cost 1), copy (cost 0)

(i-1,j-1)	(i-1,j)
(i,j-1)	(i,j)

左上邻居

Levenshtein距离: 例子

Copy Replace	delete
insert	*MIN*

- 将矩阵元素[i,j]表示为2*2的单元

		f		a		s		t	
	0	**1**	**1**	**2**	**2**	**3**	**3**	**4**	**4**
c	**1** **1**	*1* *2* *2* *1*		**2** 3 **2** **2**		**3** 4 **3** **3**		**4** 5 **4** **4**	
a	**2** **2**	**2** **2** 3 **2**		*1* *3* *3* *1*		*3* *4* *2* *2*		4 5 **3** **3**	
t	**3** **3**	**3** **3** 4 **3**		3 **2** 4 **2**		*2* *3* *3* *2*		*2* *4* *3* *2*	
s	**4** **4**	**4** **4** 5 **4**		4 **3** 5 **3**		**2** 3 4 **2**		*3* *3* *3* *3*	

Levenshtein矩阵中每个单元包含4个元素

从左上角邻居到来的开销 (copy 或 replace)	从上方邻居到来的代价 (delete)
从左方邻居到来的代价 (insert)	上述三者之中最低的代价

Levenshtein距离: 例子

		f		a		s		t	
	0	**1**	**1**	**2**	**2**	**3**	**3**	**4**	**4**
c	**1** **1**	*1* *2*	*2* *1*	**2** **2**	3 **2**	**3** **3**	4 **3**	**4** **4**	5 **4**
a	**2** **2**	**2** 3	**2** **2**	*1* *3*	*3* *1*	*3* *2*	*4* *2*	4 **3**	5 **3**
t	**3** **3**	**3** 4	**3** **3**	3 4	**2** **2**	*2* *3*	*3* *2*	*2* *3*	*4* *2*
s	**4** **4**	**4** 5	**4** **4**	4 5	**3** **3**	**2** 4	3 **2**	*3* *3*	*3* *3*

Levenshtein距离: 例子

		f	a	s	t
	0	1	2	3	4
c	1	1	2	3	4
a	2	2	1	2	3
t	3	3	2	2	2
s	4	4	3	2	3

fast中的f、s、t分别用c、t、s替换，即可得到cats，所以操作数目是3，没有别的方式会得到更小的操作数目，因此编辑距离是3。

动态规划算法(Cormen et al.)

- 最优子结构: 最优的问题解决方案中包括子解决方案，及子问题的最优解决方案（两点最短路径问题最典型的解法就是动态规划算法）。
- 重叠的子解决方案Overlapping subsolutions: 子解决方案中有重叠，如果采用暴力计算方法(穷举法)，子解决方案将会被反复计算，从而使得计算开销很大。
- 编辑距离计算中的子问题: 两个前缀子串之间的编辑距离计算。

带权重的编辑距离

- 思路： 对不同的字符进行操作时权重不同
- 希望能更敏锐地捕捉到键盘输入的错误, e.g., *m* 更可能被输成 *n* 而不是 *q*

- 因此，将 *m* 替换为 *n* 的编辑距离将低于替换为*q*的距离(即前者代价小，编辑距离短)
- 也就是输入的操作代价矩阵是一个带权重的矩阵
- 对上述动态规划算法进行修改便可以处理权重计算

利用编辑距离进行拼写校正

- 给定查询词，穷举词汇表中和该查询的编辑距离(或带权重的编辑距离)低于某个预定值的所有单词
- 求上述结果和给定的某个“正确”词表之间的交集
- 将交集结果推荐给用户
- 代价很大，实际当中往往通过启发式策略提高查找效率(如：首字母相同；保证两者之间具有较长公共子串)

课堂练习

1. 给出计算*OSLO – SNOW*之间Levenshtein距离的距离矩阵
2. 将*cat转*换成*catcat*需要哪几步Levenshtein编辑操作？

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 1				
s	2 2				
l	3 3				
o	4 4				

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 / 1	1 2 / 2 ?			
s	2 / 2				
l	3 / 3				
o	4 / 4				

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 / 1	1 2 / 2 1			
s	2 / 2				
l	3 / 3				
o	4 / 4				

		s	n	o	w
	0	**1** **1**	**2** **2**	**3** **3**	**4** **4**
o	**1** / **1**	**1** 2 / 2 **1**	2 3 / 2 ?		
s	**2** / **2**				
l	**3** / **3**				
o	**4** / **4**				

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 1	1 2 2 1	2 3 2 2		
s	2 2				
l	3 3				
o	4 4				

		s	n	o	w
	0	**1** **1**	**2** **2**	**3** **3**	**4** **4**
o	**1** **1**	**1** 2 / 2 **1**	**2** 3 / **2** **2**	2 4 / 3 ?	
s	**2** **2**				
l	**3** **3**				
o	**4** **4**				

		s	n	o	w
	0	1 1	2 2	3 3	4 4
o	1 1	1 2 / 2 1	2 3 / 2 2	2 4 / 3 2	
s	2 2				
l	3 3				
o	4 4				

		s	n	o	w
	0	**1** **1**	**2** **2**	**3** **3**	**4** **4**
o	**1** **1**	**1** 2 / 2 **1**	**2** 3 / **2** **2**	**2** 4 / 3 **2**	4 5 / 3 ?
s	**2** **2**				
l	**3** **3**				
o	**4** **4**				

		s	n	o	w
	0	**1** **1**	**2** **2**	**3** **3**	**4** **4**
o	**1** **1**	**1** 2 / 2 **1**	**2** 3 / **2** **2**	**2** 4 / 3 **2**	4 5 / **3** **3**
s	**2** **2**				
l	**3** **3**				
o	**4** **4**				

		s	n	o	w
	0	**1** **1**	**2** **2**	**3** **3**	**4** **4**
o	**1** **1**	**1** 2 2 **1**	**2** 3 **2** **2**	**2** 4 3 **2**	4 5 **3** **3**
s	**2** **2**	1 2 3 ?			
l	**3** **3**				
o	**4** **4**				

		s	n	o	w
	0	**1** **1**	**2** **2**	**3** **3**	**4** **4**
o	**1** **1**	**1** 2 / 2 **1**	**2** 3 / **2** **2**	**2** 4 / 3 **2**	4 5 / **3** **3**
s	**2** **2**	**1** 2 / 3 **1**			
l	**3** **3**				
o	**4** **4**				

		s	n	o	w
	0	**1** **1**	**2** **2**	**3** **3**	**4** **4**
o	**1** **1**	**1** 2 2 **1**	**2** 3 **2** **2**	**2** 4 3 **2**	4 5 **3** **3**
s	**2** **2**	**1** 2 3 **1**	2 3 2 ?		
l	**3** **3**				
o	**4** **4**				

		s	n	o	w
	0	**1** **1**	**2** **2**	**3** **3**	**4** **4**
o	**1** **1**	**1** 2 / 2 **1**	**2** 3 / **2** **2**	**2** 4 / 3 **2**	4 5 / **3** **3**
s	**2** **2**	**1** 2 / 3 **1**	**2** 3 / **2** **2**		
l	**3** **3**				
o	**4** **4**				

		s	n	o	w
	0	**1** **1**	**2** **2**	**3** **3**	**4** **4**
o	**1** **1**	**1** 2 2 **1**	**2** 3 **2** **2**	**2** 4 3 **2**	4 5 **3** **3**
s	**2** **2**	**1** 2 3 **1**	**2** 3 **2** **2**	3 3 3 ?	
l	**3** **3**				
o	**4** **4**				

		s	n	o	w
	0	**1** **1**	**2** **2**	**3** **3**	**4** **4**
o	**1** **1**	**1** 2 / 2 **1**	**2** 3 / **2** **2**	**2** 4 / 3 **2**	4 5 / **3** **3**
s	**2** **2**	**1** 2 / 3 **1**	**2** 3 / **2** **2**	**3** **3** / **3** **3**	
l	**3** **3**				
o	**4** **4**				

		s	n	o	w
	0	**1** **1**	**2** **2**	**3** **3**	**4** **4**
o	**1** **1**	**1** 2 2 **1**	**2** 3 **2** **2**	**2** 4 3 **2**	4 5 **3** **3**
s	**2** **2**	**1** 2 3 **1**	**2** 3 **2** **2**	**3** **3** **3** **3**	3 4 4 ?
l	**3** **3**				
o	**4** **4**				

		s	n	o	w
	0	**1** **1**	**2** **2**	**3** **3**	**4** **4**
o	**1** **1**	**1** 2 2 **1**	**2** 3 **2** **2**	**2** 4 3 **2**	4 5 **3** **3**
s	**2** **2**	**1** 2 3 **1**	**2** 3 **2** **2**	**3** **3** **3** **3**	**3** 4 4 **3**
l	**3** **3**				
o	**4** **4**				

		s	n	o	w
	0	**1** **1**	**2** **2**	**3** **3**	**4** **4**
o	**1** **1**	**1** 2 2 **1**	**2** 3 **2** **2**	**2** 4 3 **2**	4 5 **3** **3**
s	**2** **2**	**1** 2 3 **1**	**2** 3 **2** **2**	**3** **3** **3** **3**	**3** 4 4 **3**
l	**3** **3**	3 2 4 ?			
o	**4** **4**				

		s	n	o	w
	0	**1** **1**	**2** **2**	**3** **3**	**4** **4**
o	**1** **1**	**1** 2 2 **1**	**2** 3 **2** **2**	**2** 4 3 **2**	4 5 **3** **3**
s	**2** **2**	**1** 2 3 **1**	**2** 3 **2** **2**	**3** **3** **3** **3**	**3** 4 4 **3**
l	**3** **3**	3 **2** 4 **2**			
o	**4** **4**				

		s	n	o	w
	0	**1** **1**	**2** **2**	**3** **3**	**4** **4**
o	**1** **1**	**1** 2 2 **1**	**2** 3 **2** **2**	**2** 4 3 **2**	4 5 **3** **3**
s	**2** **2**	**1** 2 3 **1**	**2** 3 **2** **2**	**3** **3** **3** **3**	**3** 4 4 **3**
l	**3** **3**	3 **2** 4 **2**	2 3 3 ?		
o	**4** **4**				

		s	n	o	w
	0	**1** **1**	**2** **2**	**3** **3**	**4** **4**
o	**1** / **1**	**1** 2 / 2 **1**	**2** 3 / **2** **2**	**2** 4 / 3 **2**	4 5 / **3** **3**
s	**2** / **2**	**1** 2 / 3 **1**	**2** 3 / **2** **2**	**3** **3** / **3** **3**	**3** 4 / 4 **3**
l	**3** / **3**	3 **2** / 4 **2**	**2** 3 / 3 **2**		
o	**4** / **4**				

		s	n	o	w
	0	**1** **1**	**2** **2**	**3** **3**	**4** **4**
o	**1** **1**	**1** 2 2 **1**	**2** 3 **2** **2**	**2** 4 3 **2**	4 5 **3** **3**
s	**2** **2**	**1** 2 3 **1**	**2** 3 **2** **2**	**3** **3** **3** **3**	**3** 4 4 **3**
l	**3** **3**	3 **2** 4 **2**	**2** 3 3 **2**	3 4 3 ?	
o	**4** **4**				

		s		n		o		w	
	0	**1**	**1**	**2**	**2**	**3**	**3**	**4**	**4**
o	**1** **1**	**1** 2	2 **1**	**2** **2**	3 **2**	**2** 3	4 **2**	4 **3**	5 **3**
s	**2** **2**	**1** 3	2 **1**	**2** **2**	3 **2**	**3** **3**	3 **3**	**3** 4	4 **3**
l	**3** **3**	3 4	**2** **2**	**2** 3	3 **2**	**3** **3**	4 **3**		
o	**4** **4**								

		s	n	o	w
	0	**1** **1**	**2** **2**	**3** **3**	**4** **4**
o	**1** / **1**	**1** 2 / 2 **1**	**2** 3 / **2** **2**	**2** 4 / 3 **2**	4 5 / **3** **3**
s	**2** / **2**	**1** 2 / 3 **1**	**2** 3 / **2** **2**	**3** **3** / **3** **3**	**3** 4 / 4 **3**
l	**3** / **3**	3 **2** / 4 **2**	**2** 3 / 3 **2**	**3** 4 / **3** **3**	4 4 / 4 ?
o	**4** / **4**				

		s	n	o	w
	0	**1** **1**	**2** **2**	**3** **3**	**4** **4**
o	**1** **1**	**1** 2 / 2 **1**	**2** 3 / **2** **2**	**2** 4 / 3 **2**	4 5 / **3** **3**
s	**2** **2**	**1** 2 / 3 **1**	**2** 3 / **2** **2**	**3** **3** / **3** **3**	**3** 4 / 4 **3**
l	**3** **3**	3 **2** / 4 **2**	**2** 3 / 3 **2**	**3** 4 / **3** **3**	**4** **4** / **4** **4**
o	**4** **4**				

		s	n	o	w
	0	**1** **1**	**2** **2**	**3** **3**	**4** **4**
o	**1** **1**	**1** 2 / 2 **1**	**2** 3 / **2** **2**	**2** 4 / 3 **2**	4 5 / **3** **3**
s	**2** **2**	**1** 2 / 3 **1**	**2** 3 / **2** **2**	**3** **3** / **3** **3**	**3** 4 / 4 **3**
l	**3** **3**	3 **2** / 4 **2**	**2** 3 / 3 **2**	**3** 4 / **3** **3**	**4** **4** / **4** **4**
o	**4** **4**	4 3 / 5 ?			

		s	n	o	w
	0	**1** **1**	**2** **2**	**3** **3**	**4** **4**
o	**1** **1**	**1** 2 2 **1**	**2** 3 **2** **2**	**2** 4 3 **2**	4 5 **3** **3**
s	**2** **2**	**1** 2 3 **1**	**2** 3 **2** **2**	**3** **3** **3** **3**	**3** 4 4 **3**
l	**3** **3**	3 **2** 4 **2**	**2** 3 3 **2**	**3** 4 **3** **3**	**4** **4** **4** **4**
o	**4** **4**	4 **3** 5 **3**			

		s	n	o	w
	0	**1** **1**	**2** **2**	**3** **3**	**4** **4**
o	**1** **1**	**1** 2 2 **1**	**2** 3 **2** **2**	**2** 4 3 **2**	4 5 **3** **3**
s	**2** **2**	**1** 2 3 **1**	**2** 3 **2** **2**	**3** **3** **3** **3**	**3** 4 4 **3**
l	**3** **3**	3 **2** 4 **2**	**2** 3 3 **2**	**3** 4 **3** **3**	**4** **4** **4** **4**
o	**4** **4**	4 **3** 5 **3**	3 3 4 ?		

		s		n		o		w	
	0	**1**	**1**	**2**	**2**	**3**	**3**	**4**	**4**
o	**1** **1**	**1** 2	2 **1**	**2** **2**	3 **2**	**2** 3	4 **2**	4 **3**	5 **3**
s	**2** **2**	**1** 3	2 **1**	**2** **2**	3 **2**	**3** **3**	**3** **3**	**3** 4	4 **3**
l	**3** **3**	3 4	**2** **2**	**2** 3	3 **2**	**3** **3**	4 **3**	**4** **4**	**4** **4**
o	**4** **4**	4 5	**3** **3**	**3** 4	**3** **3**				

		s		n		o		w	
	0	**1**	**1**	**2**	**2**	**3**	**3**	**4**	**4**
o	**1** **1**	**1** 2	2 **1**	**2** **2**	3 **2**	**2** 3	4 **2**	4 **3**	5 **3**
s	**2** **2**	**1** 3	2 **1**	**2** **2**	3 **2**	**3** **3**	**3** **3**	**3** 4	4 **3**
l	**3** **3**	3 4	**2** **2**	**2** 3	3 **2**	**3** **3**	4 **3**	**4** **4**	**4** **4**
o	**4** **4**	4 5	**3** **3**	**3** 4	**3** **3**	2 4	4 ?		

		s		n		o		w	
	0	**1**	**1**	**2**	**2**	**3**	**3**	**4**	**4**
o	**1** **1**	**1** 2	2 **1**	**2** **2**	3 **2**	**2** 3	4 **2**	4 **3**	5 **3**
s	**2** **2**	**1** 3	2 **1**	**2** **2**	3 **2**	**3** **3**	**3** **3**	**3** 4	4 **3**
l	**3** **3**	3 4	**2** **2**	**2** 3	3 **2**	**3** **3**	4 **3**	**4** **4**	**4** **4**
o	**4** **4**	4 5	**3** **3**	**3** 4	**3** **3**	**2** 4	4 **2**		

		s	n	o	w
	0	**1** **1**	**2** **2**	**3** **3**	**4** **4**
o	**1** **1**	**1** 2 2 **1**	**2** 3 **2** **2**	**2** 4 3 **2**	4 5 **3** **3**
s	**2** **2**	**1** 2 3 **1**	**2** 3 **2** **2**	**3** **3** **3** **3**	**3** 4 4 **3**
l	**3** **3**	3 **2** 4 **2**	**2** 3 3 **2**	**3** 4 **3** **3**	**4** **4** **4** **4**
o	**4** **4**	4 **3** 5 **3**	**3** **3** 4 **3**	**2** 4 4 **2**	4 5 3 ?

		s	n	o	w
	0	**1** **1**	**2** **2**	**3** **3**	**4** **4**
o	**1** **1**	**1** 2 2 **1**	**2** 3 **2** **2**	**2** 4 3 **2**	4 5 **3** **3**
s	**2** **2**	**1** 2 3 **1**	**2** 3 **2** **2**	**3** **3** **3** **3**	**3** 4 4 **3**
l	**3** **3**	3 **2** 4 **2**	**2** 3 3 **2**	**3** 4 **3** **3**	**4** **4** **4** **4**
o	**4** **4**	4 **3** 5 **3**	**3** **3** 4 **3**	**2** 4 4 **2**	4 5 **3** **3**

		s	n	o	w
	0	**1** **1**	**2** **2**	**3** **3**	**4** **4**
o	**1** **1**	**1** 2 2 **1**	**2** 3 **2** **2**	**2** 4 3 **2**	4 5 **3** **3**
s	**2** **2**	**1** 2 3 **1**	**2** 3 **2** **2**	**3** **3** **3** **3**	**3** 4 4 **3**
l	**3** **3**	3 **2** 4 **2**	**2** 3 3 **2**	**3** 4 **3** **3**	**4** **4** **4** **4**
o	**4** **4**	4 **3** 5 **3**	**3** **3** 4 **3**	2 4 4 **2**	4 5 **3** **3**

		s	n	o	w
	0	**1** **1**	**2** **2**	**3** **3**	**4** **4**
o	**1** **1**	**1** 2 2 **1**	**2** 3 **2** **2**	**2** 4 3 **2**	4 5 **3** **3**
s	**2** **2**	**1** 2 3 **1**	**2** 3 **2** **2**	**3** **3** **3** **3**	**3** 4 4 **3**
l	**3** **3**	3 **2** 4 **2**	**2** 3 3 **2**	**3** 4 **3** **3**	**4** **4** **4** **4**
o	**4** **4**	4 **3** 5 **3**	**3** **3** 4 **3**	**2** 4 4 **2**	4 5 **3** **3**

如何从上述矩阵中找到编辑操作的路径？

		s	n	o	w
	0	**1** **1**	**2** **2**	**3** **3**	**4** **4**
o	**1** **1**	**1** 2 2 **1**	**2** 3 **2** **2**	**2** 4 3 **2**	4 5 **3** **3**
s	**2** **2**	**1** 2 3 **1**	**2** 3 **2** **2**	**3** **3** **3** **3**	**3** 4 4 **3**
l	**3** **3**	3 **2** 4 **2**	**2** 3 3 **2**	**3** 4 **3** **3**	**4** **4** **4** **4**
o	**4** **4**	4 **3** 5 **3**	**3** **3** 4 **3**	**2** 4 4 **2**	4 5 **3** **3**

cost	operation	input	output
1	insert	*	w

		s	n	o	w
	0	**1** **1**	**2** **2**	**3** **3**	**4** **4**
o	**1** **1**	**1** 2 / 2 **1**	**2** 3 / **2** **2**	**2** 4 / 3 **2**	4 5 / **3** **3**
s	**2** **2**	**1** 2 / 3 **1**	**2** 3 / **2** **2**	**3** **3** / **3** **3**	**3** 4 / 4 **3**
l	**3** **3**	3 **2** / 4 **2**	**2** 3 / 3 **2**	**3** 4 / **3** **3**	**4** **4** / **4** **4**
o	**4** **4**	4 **3** / 5 **3**	**3** **3** / 4 **3**	2 4 / 4 2	4 5 / 3 3

cost	operation	input	output
0	(copy)	o	o
1	insert	*	w

		s	n	o	w
	0	**1** **1**	**2** **2**	**3** **3**	**4** **4**
o	**1** **1**	**1** 2 / 2 **1**	**2** **2** / **2** **2**	**2** 3 / 3 **2**	4 5 / **3** **3**
s	**2** **2**	**1** 2 / 3 **1**	**2** **2** / **2** **2**	**3** **3** / **3** **3**	**3** 4 / 4 **3**
l	**3** **3**	3 **2** / 4 **2**	2 3 / 3 2	**3** 4 / **3** **3**	**4** **4** / **4** **4**
o	**4** **4**	4 **3** / 5 **3**	**3** **3** / 4 **3**	2 4 / 4 2	4 5 / 3 3

cost	operation	input	output
1	replace	l	n
0	(copy)	o	o
1	insert	*	w

		s	n	o	w
	0	**1** **1**	**2** **2**	**3** **3**	**4** **4**
o	**1** **1**	**1** 2 / 2 **1**	**2** 3 / **2** **2**	**2** 4 / 3 **2**	4 5 / **3** **3**
s	**2** **2**	**1** 2 / 3 **1**	**2** 3 / **2** **2**	**3** **3** / **3** **3**	**3** 4 / 4 **3**
l	**3** **3**	3 **2** / 4 **2**	2 3 / 3 **2**	**3** 4 / **3** **3**	**4** **4** / **4** **4**
o	**4** **4**	4 **3** / 5 **3**	**3** **3** / 4 **3**	2 4 / 4 **2**	4 5 / **3** **3**

cost	operation	input	output
0	(copy)	s	s
1	replace	l	n
0	(copy)	o	o
1	insert	*	w

		s	n	o	w
	0	**1** **1**	**2** **2**	**3** **3**	**4** **4**
o	**1** / **1**	**1** 2 / 2 **1**	**2** 3 / **2** **2**	**2** 4 / 3 **2**	4 5 / **3** **3**
s	**2** / **2**	**1** 2 / 3 **1**	**2** 3 / **2** **2**	**3** **3** / **3** **3**	**3** 4 / 4 **3**
l	**3** / **3**	3 **2** / 4 **2**	**2** 3 / 3 **2**	**3** 4 / **3** **3**	**4** **4** / **4** **4**
o	**4** / **4**	4 **3** / 5 **3**	**3** **3** / 4 **3**	**2** 4 / 4 **2**	4 5 / **3** **3**

cost	operation	input	output
1	delete	o	*
0	(copy)	s	s
1	replace	l	n
0	(copy)	o	o
1	insert	*	w

从*cat*到*catcat*

		c	a	t	c	a	t
	0	**1** **1**	**2** **2**	**3** **3**	**4** **4**	**5** **5**	**6** **6**
c	**1** **1**	**0** 2 2 **0**	2 3 **1** **1**	3 4 **2** **2**	**3** 5 **3** **3**	5 6 **4** **4**	6 7 **5** **5**
a	**2** **2**	2 **1** 3 **1**	**0** 2 2 **0**	2 3 **1** **1**	3 4 **2** **2**	**3** 5 **3** **3**	5 6 **4** **4**
t	**3** **3**	3 **2** 4 **2**	2 **1** 3 **1**	**0** 2 2 **0**	2 3 **1** **1**	3 4 **2** **2**	**3** 5 **3** **3**

		c	a	t	c	a	t
	0	**1** **1**	**2** **2**	**3** **3**	**4** **4**	**5** **5**	**6** **6**
c	**1** **1**	**0** 2 2 **0**	2 3 **1** **1**	3 4 **2** **2**	**3** 5 **3** **3**	5 6 **4** **4**	6 7 **5** **5**
a	**2** **2**	2 **1** 3 **1**	**0** 2 2 **0**	2 3 **1** **1**	3 4 **2** **2**	**3** 5 **3** **3**	5 6 **4** **4**
t	**3** **3**	3 **2** 4 **2**	2 **1** 3 **1**	**0** 2 2 **0**	2 3 **1** **1**	3 4 **2** **2**	**3** 5 **3** **3**

cost	operation	input	output
1	insert	*	c
1	insert	*	a
1	insert	*	t
0	(copy)	c	c
0	(copy)	a	a
0	(copy)	t	t

		c	a	t	c	a	t
	0	**1** **1**	**2** **2**	**3** **3**	**4** **4**	**5** **5**	**6** **6**
c	**1** **1**	**0** 2 / 2 **0**	2 3 / **1** **1**	3 4 / **2** **2**	**3** 5 / **3** **3**	5 6 / **4** **4**	6 7 / **5** **5**
a	**2** **2**	2 **1** / 3 **1**	**0** 2 / 2 **0**	2 3 / **1** **1**	3 4 / **2** **2**	**3** 5 / **3** **3**	5 6 / **4** **4**
t	**3** **3**	3 **2** / 4 **2**	2 **1** / 3 **1**	**0** 2 / 2 **0**	2 3 / **1** **1**	3 4 / **2** **2**	**3** 5 / **3** **3**

cost	operation	input	output
0	(copy)	c	c
1	insert	*	a
1	insert	*	t
1	insert	*	c
0	(copy)	a	a
0	(copy)	t	t

		c	a	t	c	a	t
	0	**1** **1**	**2** **2**	**3** **3**	**4** **4**	**5** **5**	**6** **6**
c	**1** / **1**	**0** 2 / 2 **0**	2 3 / **1** **1**	3 4 / **2** **2**	**3** 5 / **3** **3**	5 6 / **4** **4**	6 7 / **5** **5**
a	**2** / **2**	2 **1** / 3 **1**	**0** 2 / 2 **0**	2 3 / **1** **1**	3 4 / **2** **2**	**3** 5 / **3** **3**	5 6 / **4** **4**
t	**3** / **3**	3 **2** / 4 **2**	2 **1** / 3 **1**	**0** 2 / 2 **0**	2 3 / **1** **1**	3 4 / **2** **2**	**3** 5 / **3** **3**

cost	operation	input	output
0	(copy)	c	c
0	(copy)	a	a
1	insert	*	t
1	insert	*	c
1	insert	*	a
0	(copy)	t	t

		c	a	t	c	a	t
	0	**1** **1**	**2** **2**	**3** **3**	**4** **4**	**5** **5**	**6** **6**
c	**1** **1**	**0** 2 2 **0**	2 3 **1** **1**	3 4 **2** **2**	**3** 5 **3** **3**	5 6 **4** **4**	6 7 **5** **5**
a	**2** **2**	2 **1** 3 **1**	**0** 2 2 **0**	2 3 **1** **1**	3 4 **2** **2**	**3** 5 **3** **3**	5 6 **4** **4**
t	**3** **3**	3 **2** 4 **2**	2 **1** 3 **1**	**0** 2 2 **0**	2 3 **1** **1**	3 4 **2** **2**	**3** 5 **3** **3**

cost	operation	input	output
0	(copy)	c	c
0	(copy)	a	a
0	(copy)	t	t
1	insert	*	c
1	insert	*	a
1	insert	*	t

提纲

- 上一讲回顾
- 词典
- 通配查询
- 编辑距离
- 拼写校正
- Soundex

拼写校正

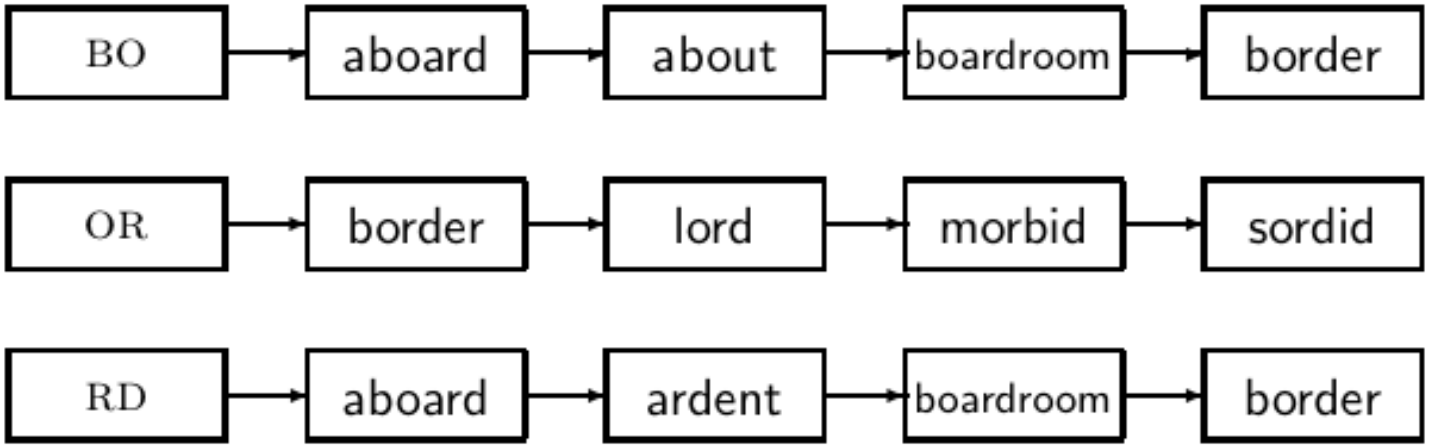
- 刚才已经介绍如何利用编辑距离进行词独立方式下的拼写校正
- 另一种方法： *k*-gram索引
- 上下文敏感的拼写校正
- 拼写校正中的一般问题

基于*k*-gram索引的拼写校正

- 列举查询词项中的所有*k*-gram
  - 例子：采用2-gram索引，错误拼写的单词为bordroom
  - 2-gram: *bo, or, rd, dr, ro, oo, om*
- 利用 *k*-gram索引返回和能够匹配很多查询*k*-gram的正确单词
- 匹配程度(数目或者指标)上可以事先设定阈值
- E.g., 比如最多只有 3 个*k*-gram不同

2-gram索引示意图

查询“bord”的2-gram索引：

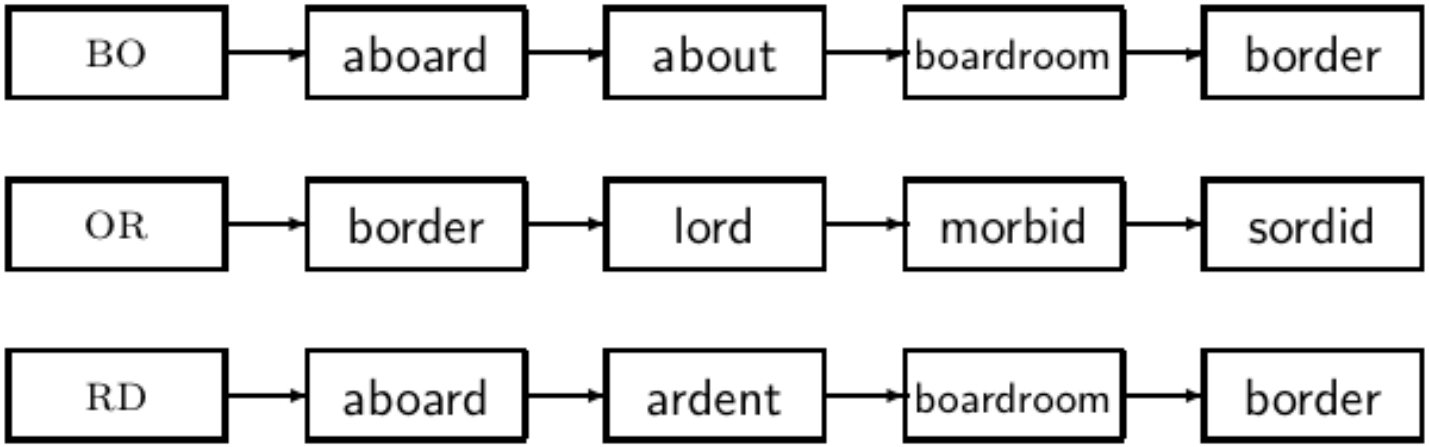


命中至少两个2-gram的词项：aboard、boardroom、border

使用Jaccard系数进行过滤，Jaccard系数超过阈值的词项才返回

2-gram索引示意图

查询“bord”的2-gram索引：



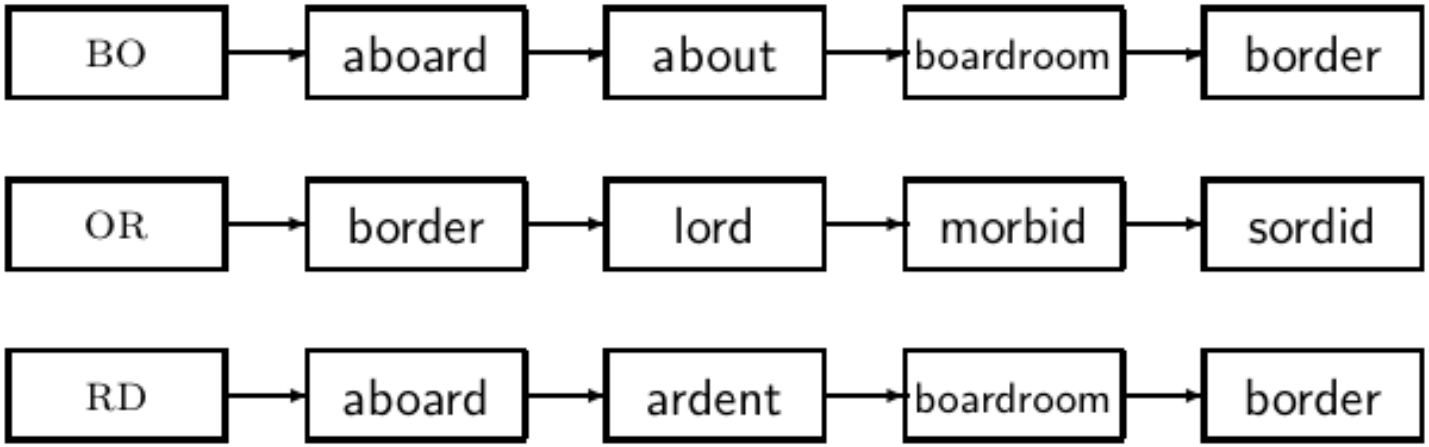
命中至少两个2-gram的词项：aboard、boardroom、border

$$J(A,B) = |A \cap B| / |A \cup B|$$

- A：查询词q的k-gram集合
- B：候选集词项t的k-gram集合

2-gram索引示意图

查询“bord”的2-gram索引：



命中至少两个2-gram的词项：aboard、boardroom、border

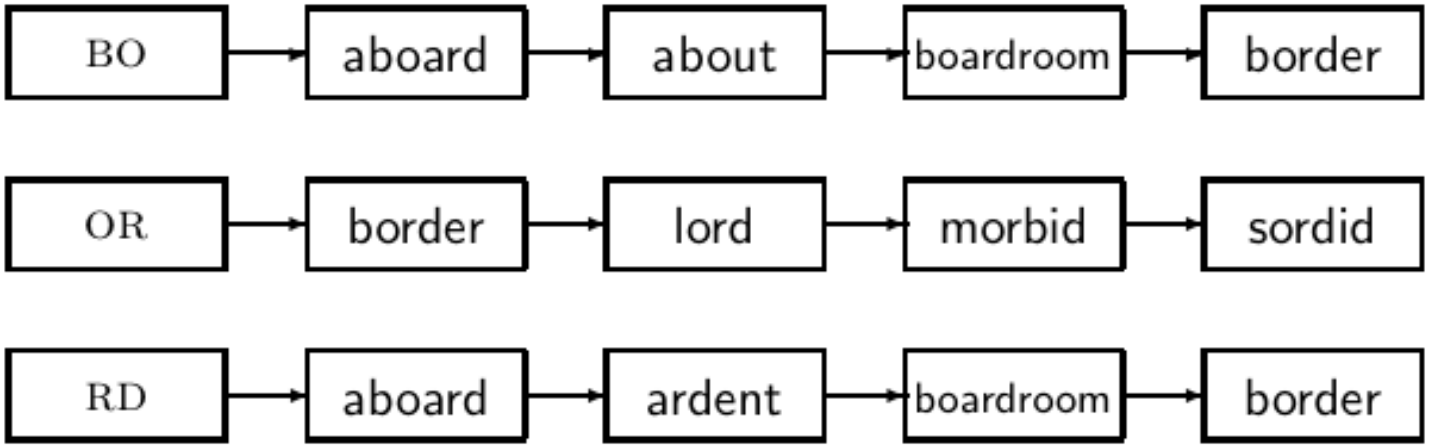
$$J(A,B)=|A\cap B|/|A\cup B|$$

q=bord, t=boardroom

$J=2/8+3-2$

2-gram索引示意图

查询“bord”的2-gram索引：



命中至少两个2-gram的词项：aboard、boardroom、border

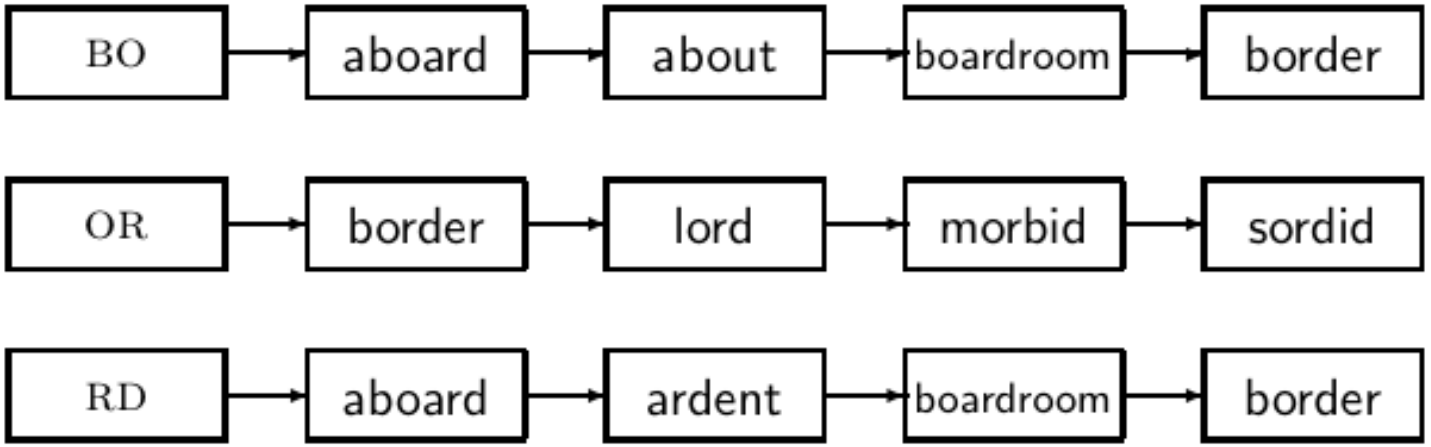
$$J(A,B)=|A\cap B|/|A\cup B|$$

q=bord, t=boardroom

$$J=2/8+3-2$$

2-gram索引示意图

查询“bord”的2-gram索引：



命中至少两个2-gram的词项：aboard、boardroom、border

$$J(A,B)=|A\cap B|/|A\cup B|$$

q=bord, t=aboard → J=?

q=bord, t=border → J=?

上下文敏感的拼写校正

- 例子: *an asteroid that fell form the sky*
- 如何对*form*纠错？
- 一种方法: 基于命中数(hit-based)的拼写校正
  - 对于每个查询词项返回 相近的“正确” 词项**【使用词项独立法】**
  - *flew form munich: flea ->flew, from -> form, munch ->munich*
  - 组合所有可能
  - 搜索 *“flea form munich”*
  - 搜索 *“flew from munich”*
  - 搜索 *“flew form munch”*
  - 正确查询 *“flew from munich”* 会有最高的结果命中数

- 假定 *flew*有7个可能的候选词, form 有20个， *munich* 有3个，那么需要穷举出多少个查询？

上下文敏感的拼写校正

- 刚才提到的基于命中数的算法效率不高
- 一种更高效的做法是：使用启发式方法，减少可能的拼写空间大小
  - 从索引文档集中，搜索高频双词
  - 从查询库(比如历史查询)中，搜索高频双词

拼写校正中的一般问题

- 用户交互界面问题
  - 全自动 vs. 推荐式校正方法(Did you mean…?)
  - 推荐式校正方法通常只给出一个建议
  - 如果有多个可能的正确拼写怎么办？
  - 平衡: 交互界面的简洁性 vs. 强大性
- 开销问题
  - 拼写校正的开销很大
  - 避免对所有查询都运行拼写校正模块
  - 只对返回结果很少的查询运行拼写校正模块
  - 猜测: 主流搜索引擎的拼写校正模块非常高效，有能力对每个查询进行拼写校正

课堂练习: Peter Norvig拼写校正工具的理解

```
import re
from collections import Counter

def words(text): return re.findall(r'\w+', text.lower())

WORDS = Counter(words(open('big.txt').read()))

def P(word, N=sum(WORDS.values())):
    "Probability of `word`."
    return WORDS[word] / N

def correction(word):
    "Most probable spelling correction for word."
    return max(candidates(word), key=P)

def candidates(word):
    "Generate possible spelling corrections for word."
    return (known([word]) or known(edits1(word)) or known(edits2(word)) or [word])

def known(words):
    "The subset of `words` that appear in the dictionary of WORDS."
    return set(w for w in words if w in WORDS)

def edits1(word):
    "All edits that are one edit away from `word`."
    letters    = 'abcdefghijklmnopqrstuvwxyz'
    splits     = [(word[:i], word[i:])    for i in range(len(word) + 1)]
    deletes    = [L + R[1:]               for L, R in splits if R]
    transposes = [L + R[1] + R[0] + R[2:] for L, R in splits if len(R)>1]
    replaces   = [L + c + R[1:]           for L, R in splits if R for c in letters]
    inserts    = [L + c + R               for L, R in splits for c in letters]
    return set(deletes + transposes + replaces + inserts)

def edits2(word):
    "All edits that are two edits away from `word`."
    return (e2 for e1 in edits1(word) for e2 in edits1(e1))
```

```
>>> correction('speling')
'spelling'

>>> correction('korrectud')
'corrected'
```

```
>>> len(edits1('somthing'))
442
```

```
>>> known(edits1('somthing'))
{'something', 'soothing'}
```

```
>>> len(set(edits2('something'))
90902

>>> known(edits2('something'))
{'seething', 'smoothing', 'something', 'soothing'}

>>> known(edits2('somthing'))  8个
{'loathing', 'nothing', 'scathing', 'seething', 'sm
```

提纲

- 上一讲回顾
- 词典
- 通配查询
- 编辑距离
- 拼写校正
- Soundex

Soundex

- 一种特殊的拼写错误(发音相似的拼写错误)比如:
  chebyshev / tchebyscheff 【切比雪夫】
- Soundex是寻找发音相似的单词的方法
- 具体算法:
  - 将词典中每个词项转换成一个4字符缩减形式
  - 对查询词项做同样的处理
  - 基于4-字符缩减形式进行索引和搜索

Soundex 算法

1. 保留词项的首字母
2. 将后续所有的A、E、I、O、U、H、W及Y等字母转换为0。
3. 按照如下方式将字母转换成数字：
   - B, F, P, V → 1
   - C, G, J, K, Q, S, X, Z → 2
   - D,T → 3
   - L → 4
   - M, N → 5
   - R → 6
4. 将连续出现的两个同一字符转换为一个字符直至再没有这种现象出现。
5. 在结果字符串中剔除0，并在结果字符串尾部补足0，然后返回前四个字符，该字符由1个字母加上3个数字组成。

例子: 采用Soundex算法处理*HERMAN*

- 保留 H
- *ERMAN* → *0RM0N*
- *0RM0N* → *06505*
- *06505* → *06505*
- *06505* → *655*
- 返回 *H655*
- 注意: *HERMANN* 会产生同样的编码

Soundex的应用情况

- 在IR中并不非常普遍
- 适用于“高召回率”任务 (e.g., 国际刑警组织Interpol在全球范围内追查罪犯)
- Zobel and Dart (1996)提出了一个更好的发音匹配方法

课堂练习

- **计算你的姓的拼音的Soundex编码**

本讲小结

- 词典的数据结构：访问效率和支持查找的方式
  - 哈希表 vs. 树结构
- 容错式检索(Tolerant retrieval)：查询词项和文档词项不匹配
  - 通配查询：包含通配符*的查询
    - 轮排索引 vs. k-gram索引
  - 拼写校正：
    - 编辑距离 vs. k-gram相似度
    - 词独立校正法 vs. 上下文敏感校正法
    - Soundex算法

前一段小结

- 目前的索引方式和查询方式
  - 布尔查询：普通倒排索引、+跳表
  - 短语查询/临近查询：双词索引、位置索引
  - 通配查询：轮排索引、k-gram索引
  - 拼写错误的查询：编辑距离、k-gram索引、+上下文敏感
  - 发音错误的查询：Soundex索引
- 包含各种格式索引的搜索引擎，可以处理如下查询
  - (SPELL(moriset) /3 toron*to) OR SOUNDEX(chaikofski)

参考资料

- 《信息检索导论》第3章、MG4.2
- 高效拼写校正方法:

K. Kukich. Techniques for automatically correcting words in text. ACM Computing Surveys 24(4), Dec 1992.
J. Zobel and P. Dart. Finding approximate matches in large lexicons. Software - practice and experience 25(3), March 1995. http://citeseer.ist.psu.edu/zobel95finding.html
Mikael Tillenius: Efficient Generation and Ranking of Spelling Error Corrections. Master's thesis at Sweden's Royal Institute of Technology. http://citeseer.ist.psu.edu/179155.html

- Soundex演示
- Levenshtein距离的演示
- Peter Norvig的拼写校正工具

课后练习

- 待补充