词项词典和倒排记录表
The term vocabulary & postings lists

目录

第一部分：如何建立词项词典？

- 文档解析(Parsing a document)
- 词条化 (Tokenization)
- 停用词 (Stop Words)
- 词项归一化 (Normalization)
- 词干还原 (Stemming)
- 词形归并 (Lemmatization)

第二部分：如何实现倒排记录表？

- 快速合并算法：带跳表的倒排记录表(Skip lists)
- 包含位置信息的倒排记录表以及短语查询

建立词项(Term)词典过程

待建索引文档集合

词条化模块
Tokenizer

词条化

Friends | Romans | Countrymen

语言学预处理模块
Linguistic modules

friend | roman | countryman

语言学预处理

索引器
Indexer

倒排记录表

friend	2 → 4 →
roman	1 → 2 →
countryman	13 → 16

文档解析(Parsing a document)

- 文档包含哪些格式？
  - pdf/word/excel/html et al.
- 文档中包含的语言？
- 文档使用何种编码方式？

上述问题都可以看成是机器学习中的分类问题，但在实际中往往采用启发式方法来实现。(后面章节讨论)

不同文档格式的识别

PDF

务中，推荐系统面临.
提出一个针对个性化推荐
们维持一个兴趣物品的配置
维持一个包括人口统计学信
置文件里的特征，我们开发
轻量计算开销的离线模型。
日新闻模块。
不是缺少物品，而是如何
的利用。
程：收集和存储网
像每个访客

DOC

在基于网络提供的动态物品服务中，推荐系统面临这样的困难：及时的提供高质量的新项目；给新用户做出推荐。我们提出一个针对个性化推荐系统的基于特征的机器学习方法，能够有效的处理冷启动问题。我们维持一个兴趣物品的配置文档，里面的物品（比如流行度，新鲜度）会实时更新。我们同样维持一个包括人口统计学信息的用户的配置文件和在雅虎属性内的活动。基于用户和物品配置文件里的特征，我们开发了一个预测性的双线性回归模型。

这个方法形成了一个具有轻量计算开销的离线模型。

数据集：雅虎头版中的今日新闻模块。

大多数组织面临的最大困难不是缺少物品，而是如何通过在恰当的时间辨认出最合适的用户，使拥有的物品得到最佳化的利用。

个性化的推荐包括这样一个过程：收集和存储网站访问者的信息，管理物品资源，分析现在和过去用户的交互行为。通过分析像每个访客传递恰当的物品。

文档中的语言

ཨཨ། སྐུ་ཞིང་བོད་དུ་ལེའུས་རྒྱར་དགའ་བསུ་ཞུ།

欢迎您到西藏来！

སྐུ་ཁམས་བཟང་།

您好！早上好！下午好！晚上好！

བཀྲ་ཤིས་བདེ་ལེགས།

吉祥如意

དགོངས་པ་མ་ཚོམ།

对不起

བྱག་རྗེ་ཆེ།

谢谢

文档中的编码方式

- 7bit ASCII？
- UNICODE？
  - UTF-8、UTF-16、UTF-32
- Email对二进制附件的编码
  - Content-Type: text/html;
  - charset="gb2312"
  - Content-Transfer-Encoding: base64

复杂因素：格式/语言

- 待索引文档集中包含不同语言的文档
  - 单独的一个索引应该包含不同语言的文档
- 一个文档或者其附件中包含多种语言或格式
  - 例子：一封法语的邮件中包含德语的pdf
- **文档单位**的选择？
  - 一个文件？
  - 一封email？
  - 一封带有5个附件的email？
  - 一组文件？

目录

第一部分：如何建立词项词典？

- 文档解析(Parsing a document)
- 词条化 (Tokenization)
- 停用词 (Stop Words)
- 词项归一化 (Normalization)
- 词干还原 (Stemming)
- 词形归并 (Lemmatization)

第二部分：如何实现倒排记录表？

- 快速合并算法：带跳表的倒排记录表
- 包含位置信息的倒排记录表以及短语查询

什么是词条化(Tokenization)

- 词条化：将给定的字符序列拆分成一系列子序列的过程，其中每一个子序列称之为一个“词条”Token。
- 输入：“*Friends, Romans and Countrymen*”
- 输出：
  - *Friends*
  - *Romans*
  - *Countrymen*
- 每个词条都作为候选的索引。
- 但是什么是有效的索引？

词条(Tokens)
词项(Terms)

词条化可能遇到的问题(英文)

e.g.： Finland’s capital →Finland? Finlands? Finland’s?

- 连字符问题？
  - Hewlett-Packard → Hewlett和Packard 是二个词条吗？
  - State-of-the-art
  - Co-education
- 空格问题？
  - San Francisco是一个词条还是二个词条？
- 连字符和空格相互影响
  - Lowercase，lower-case，lower case
- 英文句号的考虑
  - IEEE 802.3 X.25 X.509
- 数字的考虑
  - Tel:3601000, Mar.2011
  - 查询2009至2011年间车祸死亡的人数

词条化可能遇到的问题(中文)

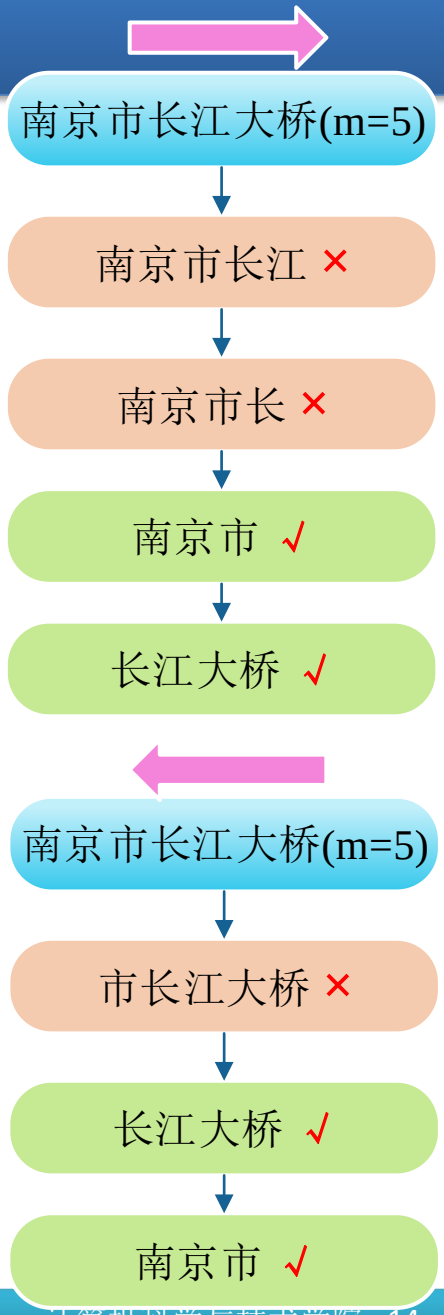
- Out of Vocabulary
  - 人名、地名、机构名、一些新词
- Ambiguity
  - 同一句子有多种可能的分词结果
    - 南京市长江大桥
      - 南京 市长 江大桥
      - 南京市 长江大桥
    - 兵乓球拍卖完了
      - 兵乓 球拍 卖完 了
      - 兵乓球 拍卖 完了

词条化的策略

- 针对不同的语言，采取不同策略的词条化方法
- 分词的基本方法：
  - 基于词典(规则)的方法
  - 按照一定策略将待分析的汉字串与一个“词典”中的词条进行匹配，如果匹配成功，那么该汉字串就是一个词。
  - 基于统计的方法
    - 训练：根据观测到的数据(人工标注好的语料)的统计特征对模型参数进行估计。
    - 分词：通过模型计算各种分词出现的概率，将概率最大的分词结果作为最终结果。

基于词典(规则)的方法

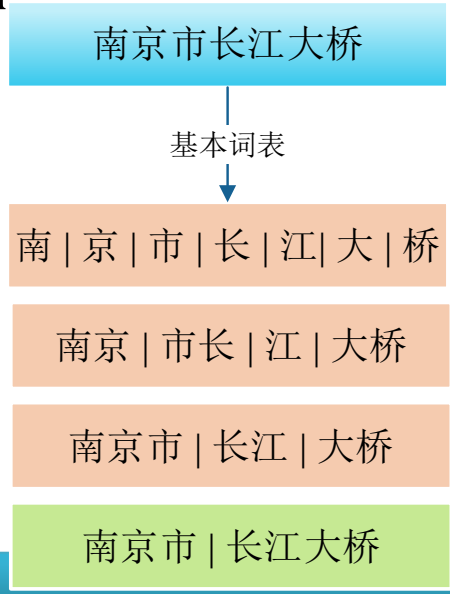
- 按照扫描方向：正向匹配和逆向匹配
- 按照扫描长度：最大匹配和最小匹配
- 正向最大匹配
  - 从左向右取待切分汉语句的*m*个字符作为匹配字段(*m*为词典中最长词条个数)；
  - 查找词典并进行匹配；
    - 若匹配成功，则将这个匹配字段作为一个词切分出来；
    - 若匹配不成功，则将这个匹配字段的最后一个字去掉，剩下的字符串作为新的匹配字段，进行再次匹配，重复以上过程，直到切分出所有词为止。
  - 例：南京市长江大桥 (*m*=5)
- 逆向最大匹配
- 双向最大匹配
  - 双向最大匹配法是将正向最大匹配法得到的分词结果和逆向最大匹配法的到的结果进行比较，把所有可能的最大词都分出来。



基于统计的方法

- *n*-gram：基于假设，第*n*个词的出现只与前面*n*-1个词相关，而与其它任何词都不相关，整句的概率就是各个词出现概率的**乘积**。
  - 一个句子$S=\{t_1 t_2 t_3 \ldots t_N\}$
  - 句子出现的概率：Y是分词序列 $Y^* = \arg\max_Y P(Y \mid S) = \prod_{i=1}^{N} P(t_i \mid t_1 \ldots t_{i-1})$
  - Unigram：$P_{\text{uni}}(Y|S) = P(t_1)P(t_2)P(t_3)P(t_4)\ldots P(t_N)$
  - Bigram：只考虑前一个词项的出现情况，

    $P_{\text{bi}}(Y|S) = P(t_1)P(t_2|t_1)P(t_3|t_2)\ldots P(t_N|t_{N-1})$
  - Trigram



Bigram示例

$P(S)=P(南)P(京|南)P(市|京)P(长|市)P(江|长)P(大|江)P(桥|大)$ ×

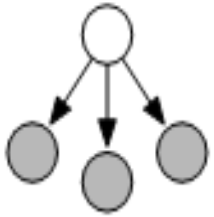
$P(S)=P(南京)P(市长|南京)P(江|市长)P(大桥|江)$ ×

$P(S)=P(南京市)P(长江|南京市)P(大桥|长江)$ ×

$P(S)=P(南京市)P(长江大桥|南京市)$ √ **Max**

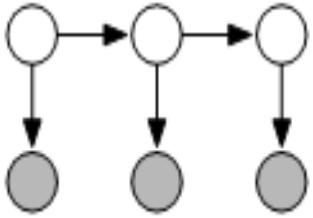
NLP中概率图模型的演变

点 线 图



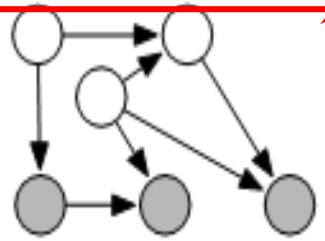
朴素贝叶斯

SEQUENCE
序列



HMMs

GENERAL GRAPHS
一般图



生成式有向图

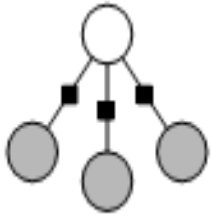
CONDITIONAL

在一定条件下

CONDITIONAL

在一定条件下

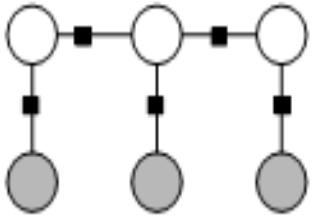
CONDITIONAL



Logistic Regression

逻辑回归

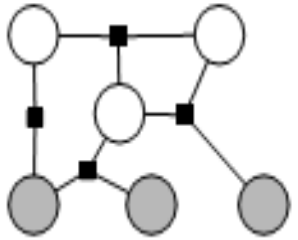
SEQUENCE
序列



Linear-chain CRFs

线性链式CRFs

GENERAL GRAPHS
一般图



General CRFs

通用CRFs

基于HMM的中文分词方法

- 给定一个观察序列(句子)$X=x_1x_2\ldots x_t\ldots x_T$, 其中$x_t$是一个字、词等文字单元
- 假设X的状态序列(如词的开始符、词的结束符等)为Y
  - $y_t(i)$有M个状态　$Y = y_1(i)y_2(i)...y_t(i)...y_T(i),\ 1 \le i \le M$

$$Y^* = \arg\max_Y P(Y \mid X) = \arg\max_Y \frac{P(Y,X)}{P(X)} \propto \arg\max_Y P(X \mid Y)P(Y)$$

1) 独立性假设 $\Rightarrow P(X \mid Y) = \prod_{t=1}^{T} P(x_t \mid y_t)$

2) 马尔可夫(一阶)假设：$P(y_t \mid y_{t-1}y_{t-2}\ldots y_1) = P(y_t \mid y_{t-1}) \Rightarrow P(Y)=P(y_1)\prod_{t=2}^{T} P(y_t \mid y_{t-1})$

$$Y^* = \arg\max_Y P(y_1)\prod_{t=1}^{T} P(x_t \mid y_t)\prod_{t=2}^{T} P(y_t \mid y_{t-1})$$

$$= \arg\max_Y P(y_1)P(x_1 \mid y_1)\prod_{t=2}^{T}[P(y_t \mid y_{t-1})P(x_t \mid y_t)]$$

$$= \arg\max_Y [P(y_1)P(x_1 \mid y_1)][P(y_2 \mid y_1)P(x_2 \mid y_2)][P(y_3 \mid y_2)P(x_3 \mid y_3)]\ldots$$

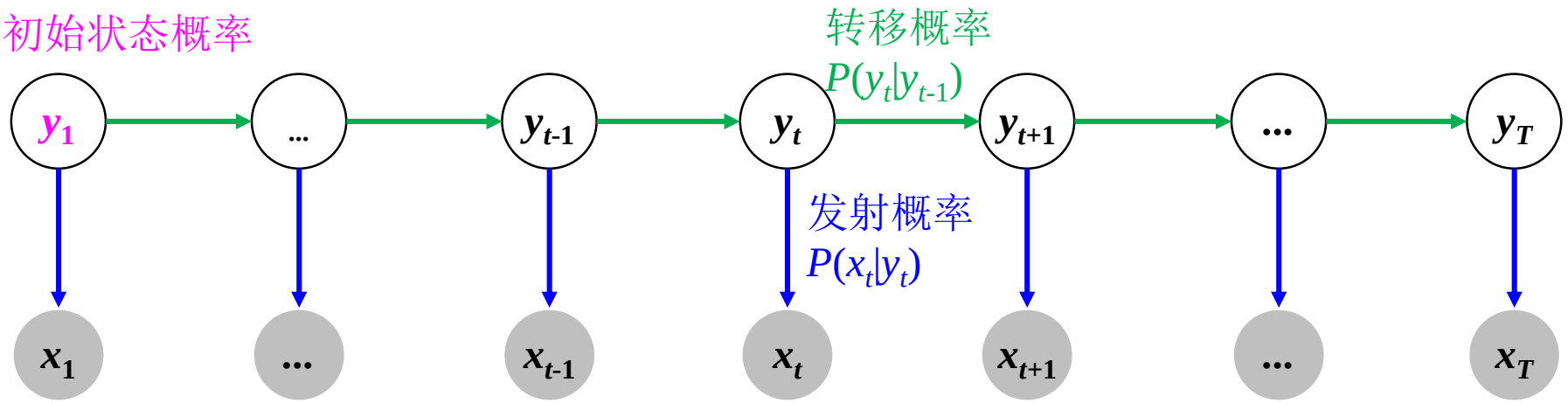
$$Y^* = \arg\max_Y P(y_1)\prod_{t=1}^{T} P(x_t \mid y_t)\prod_{t=2}^{T} P(y_t \mid y_{t-1})$$

$$= \arg\max_Y P(y_1)P(x_1 \mid y_1)\prod_{t=2}^{T}[P(y_t \mid y_{t-1})P(x_t \mid y_t)]$$

$$= \arg\max_Y [P(y_1)P(x_1 \mid y_1)][P(y_2 \mid y_1)P(x_2 \mid y_2)][P(y_3 \mid y_2)P(x_3 \mid y_3)]...$$

- 发射概率 $P(x_t|y_t)$
- 转移概率 $P(y_t|y_{t-1})$
  - Y共有M个状态 $\sum_{i=1}^{M} P(y_t(i) \mid y_{t-1}) = 1$
- 初始状态概率 $P(y_1(i))$ (i=1...M)
- HMM包括隐层状态Y，观察序列X，状态转移概率A，符号发射概率B，和初始状态概率分布π。
- HMM表示为μ={A,B, π}，参数通过训练集来学习获得。

HMM可以用有向图模型来表示，因为states (*Y*)与observations (*X*)之间存在着明显的依赖关系。



中文分词

- 输入观察序列*X*：南京市长江大桥
- 状态集合*Y*(*i*) ={B, M, E, S}：

状态*Y*	Begin	Middle	End	Single
解释	词的开始字	词的中间字	词的结束字	单字成词
示例	南京的“南”	乒乓球的“乓”	南京的“京”	你

- 输出状态序列*Y*：BMEBMME

Viterbi 搜索算法

给定模型μ和观察序列$X=x_1x_2\ldots x_t\ldots x_T$的条件下求概率最大的状态序列$Y=y_1y_2\ldots y_t\ldots y_T$：

$$Y^* = \arg\max_{Y} P(Y \mid X, \mu)$$

Viterbi 算法: 动态搜索最优状态序列。

定义：Viterbi 变量$\delta_t(y(i))$**是在时间** t **时，模型沿着某一条路径到达状态** $y(i)$**，并输出观察序列** $X=x_1x_2\ldots x_t$**的最大概率：**

$$\delta_t(y(i)) = \max_{y_1y_2\cdots y_{t-1}} P(y_1y_2\cdots y_{t-1}y_t(i), x_1x_2\cdots x_{t-1}x_t \mid \mu)$$

从状态$y_t(j)$转移到状态$y_{t+1}(i)$的概率

t+1步状态$y_{t+1}(i)$发射观察值x_{t+1}的概率

$$\delta_{t+1}(y(i)) = \max_{y(j)}\{\delta_t(y(j)) \cdot P(y_{t+1}(i) \mid y_t(j))\} \cdot P(x_{t+1} \mid y_{t+1}(i))$$

where $y(i), y(j) \in \{B, M, E, S\}$

递归计算：

算法描述

(1)初始化：$\delta_1(i) = y_1(i)P(x_1 \mid y_1(i)), \quad 1 \le i \le M$

概率最大的**路径**变量： $\psi_1(i) = 0$

(2)递推计算：

$$\delta_t(i) = \max_{1 \le j \le M}\{\delta_{t-1}(j) \cdot P(y_t(i) \mid y_{t-1}(j))\} \cdot P(x_t \mid y_t(i)), 2 \le t \le T, 1 \le j, i \le M$$

第t步从所有可能的$j \rightarrow i$的M条路径中取最大值

$$\psi_t(i) = \arg\max_{1 \le j \le M}[\delta_{t-1}(j) \cdot P(y_t(i) \mid y_{t-1}(j))] \cdot P(x_t \mid y_t(i)), \quad 2 \le t \le T, \quad 1 \le i, j \le M$$

arg是从1...t路径的累积概率最大

(3)结束：

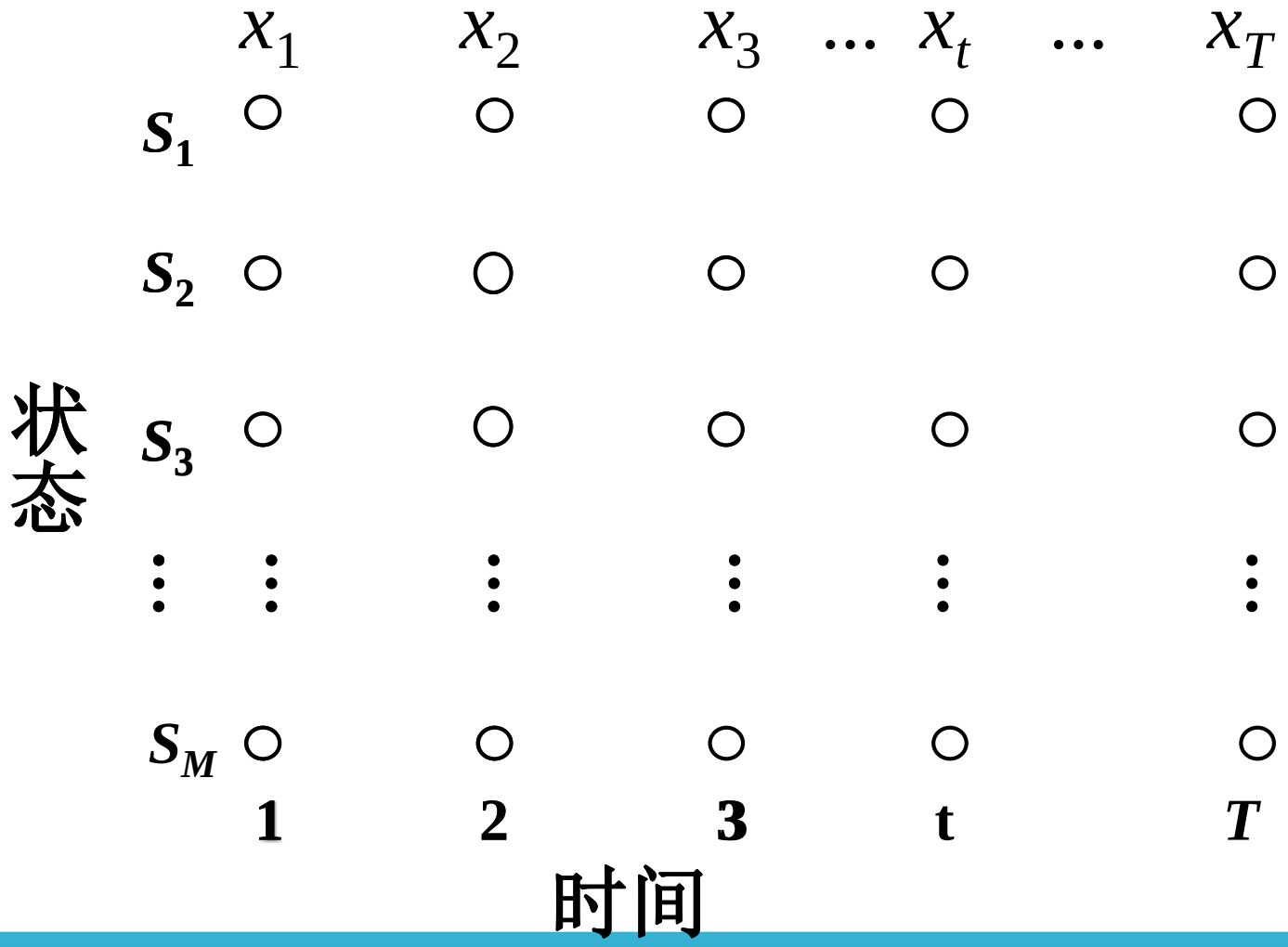
$$Y = \arg\max_{1 \le i \le M}[\delta_T(i)], \qquad P(Y) = \max_{1 \le i \le M} \delta_T(i)$$

(4)通过回溯得到路径（状态序列）：

$$y_t = \psi_{t+1}(y_{t+1}), \quad t = T-1,\ T-2,\ \cdots,\ 1$$

算法的时间复杂度：$O(M^2T)$

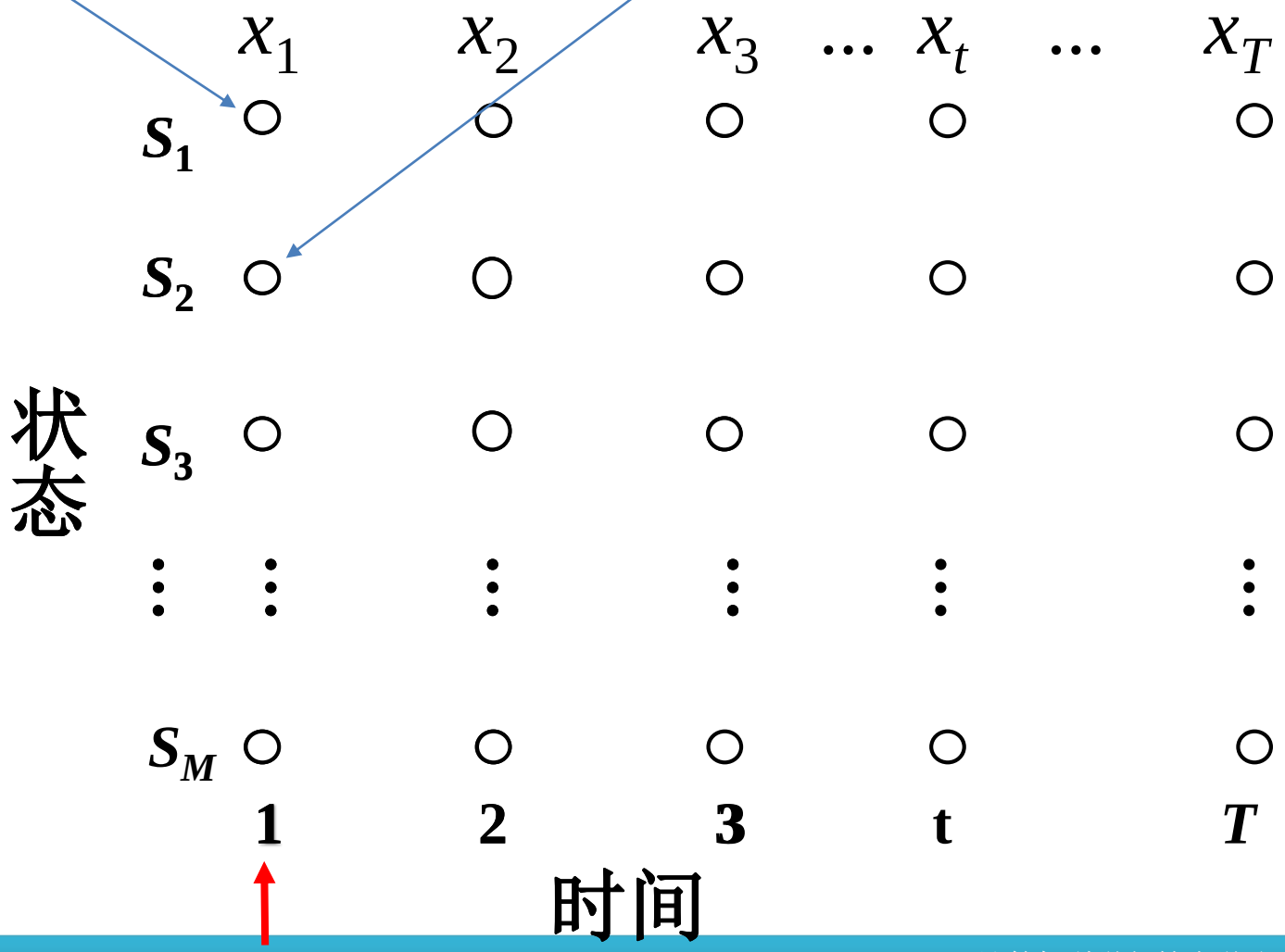
图解Viterbi搜索过程



$$\delta_1(i) = y_1(i)P(x_1 \mid y_1(i)), \quad 1 \le i \le M$$

$$\delta_1(1) = y_1(1)P(x_1 \mid y_1(1)) \quad \delta_1(2) = y_1(2)P(x_1 \mid y_1(2))$$

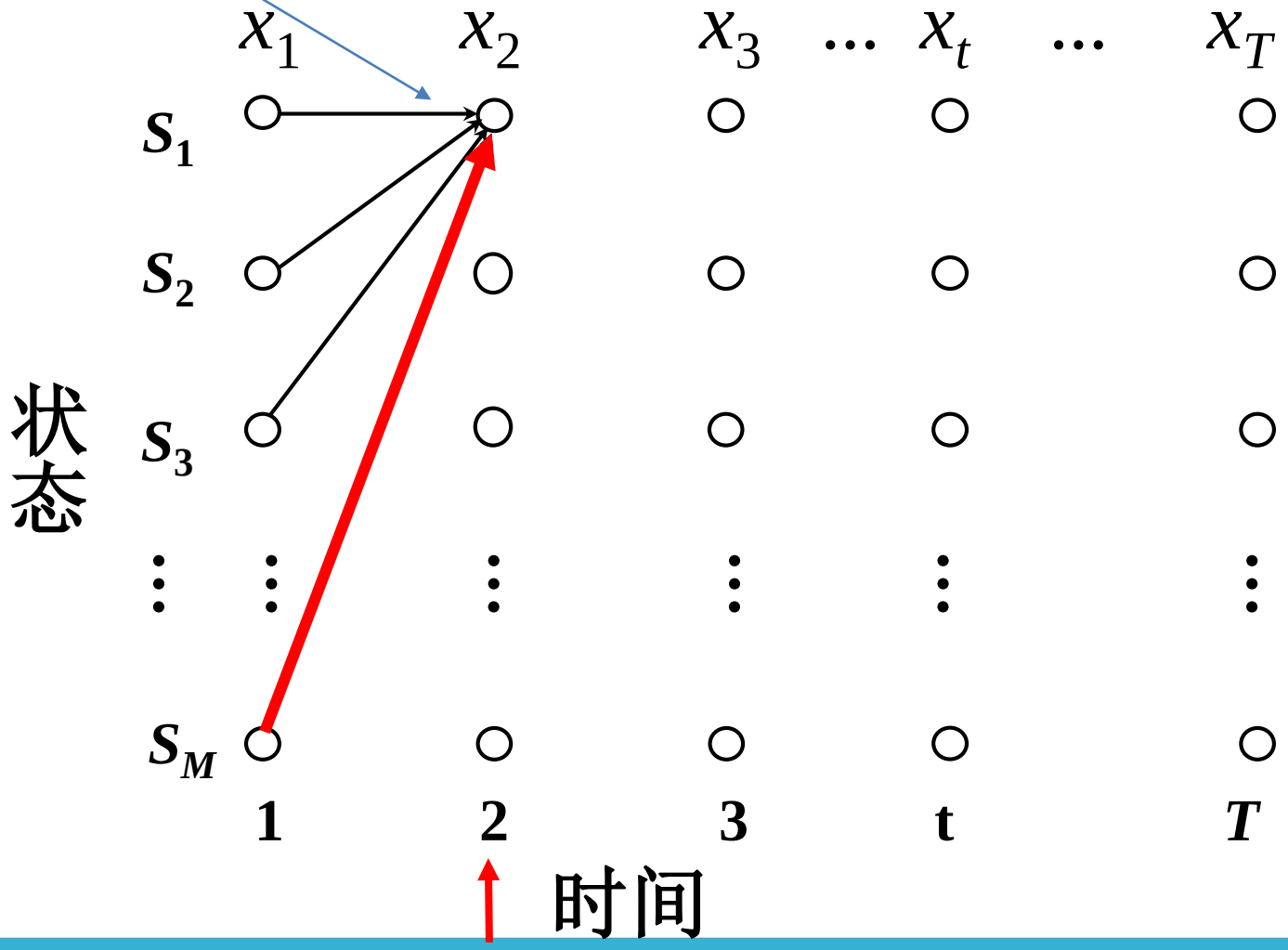
图解 Viterbi 搜索过程



$$\delta_t(i) = \max_{1 \le j \le M} \{\delta_{t-1}(j) \cdot P(y_t(i) \mid y_{t-1}(j))\} \cdot P(x_t \mid y_t(i)), 2 \le t \le T, 1 \le j, i \le M$$

$$\delta_2(1) = \max_{1 \le j \le M} \{\delta_1(j) \cdot P(y_2(1) \mid y_1(j))\} \cdot P(x_2 \mid y_2(1))$$

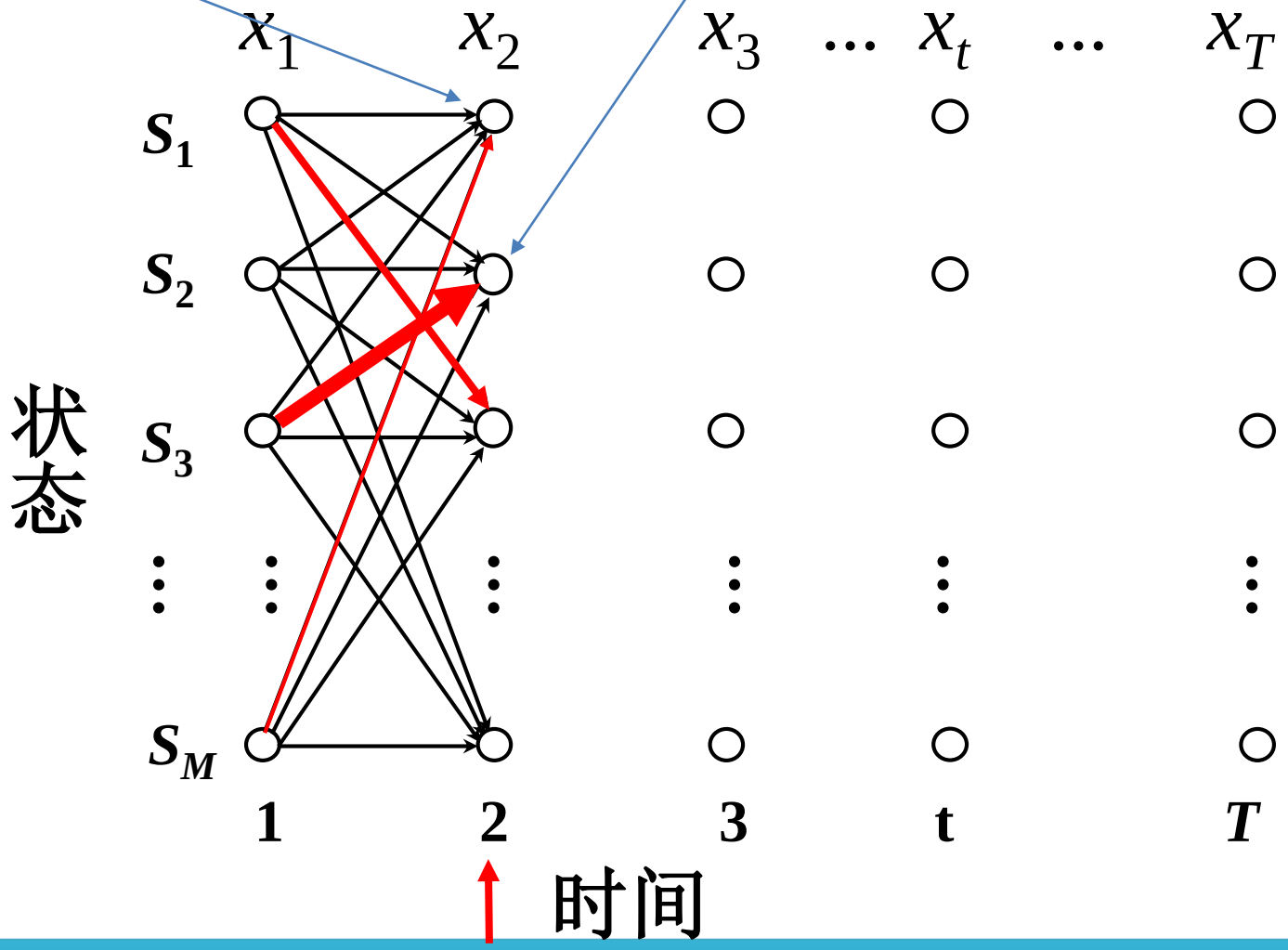
图解 Viterbi 搜索过程



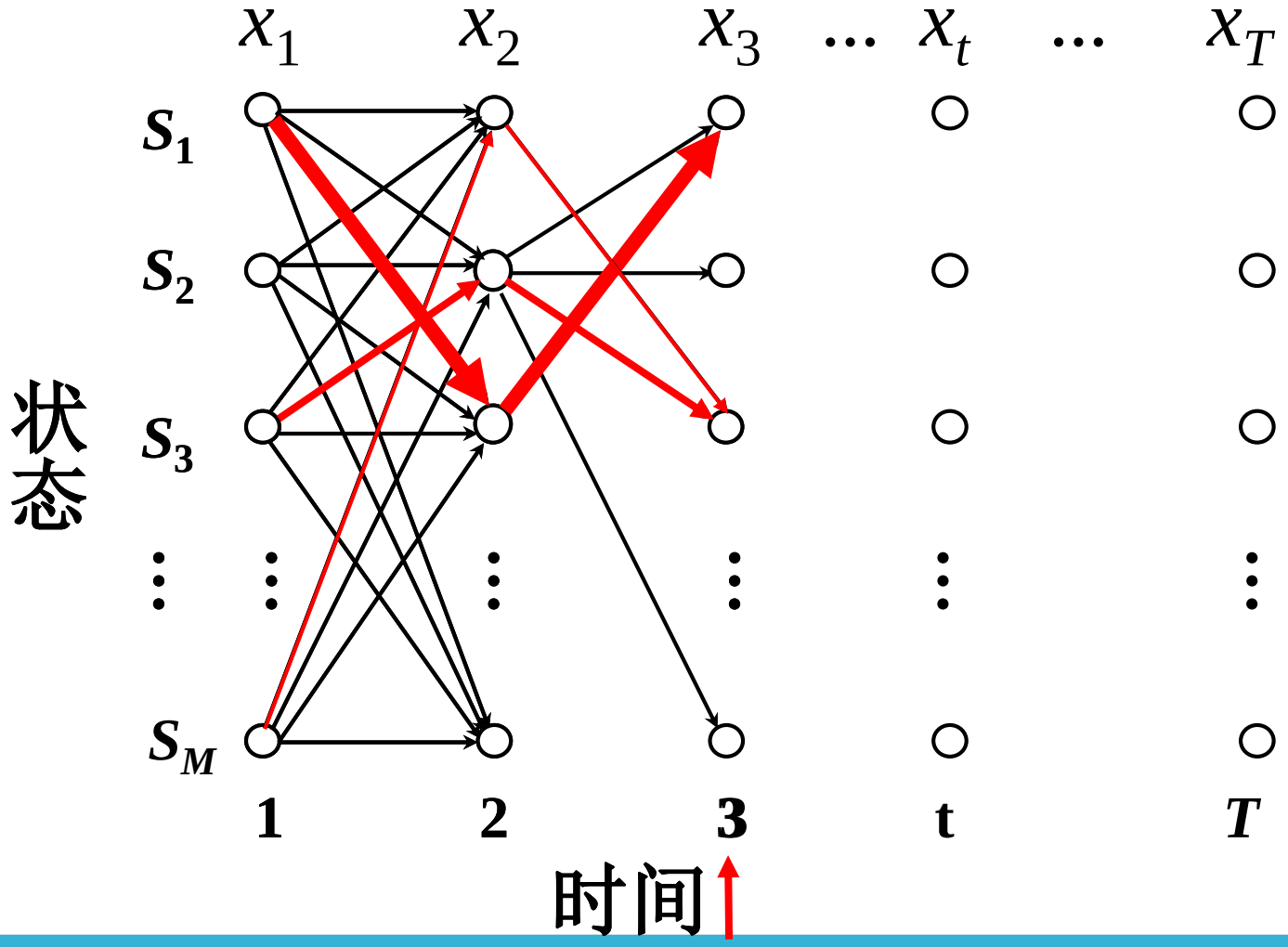
$$\delta_2(1) = \max_{1 \le j \le M} \{\delta_1(j) \cdot P(y_2(1) \mid y_1(j))\} \cdot P(x_2 \mid y_2(1))$$

$$\delta_2(2) = \max_{1 \le j \le M} \{\delta_1(j) \cdot P(y_2(2) \mid y_1(j))\} \cdot P(x_2 \mid y_2(2))$$

图解 Viterbi 搜索过程



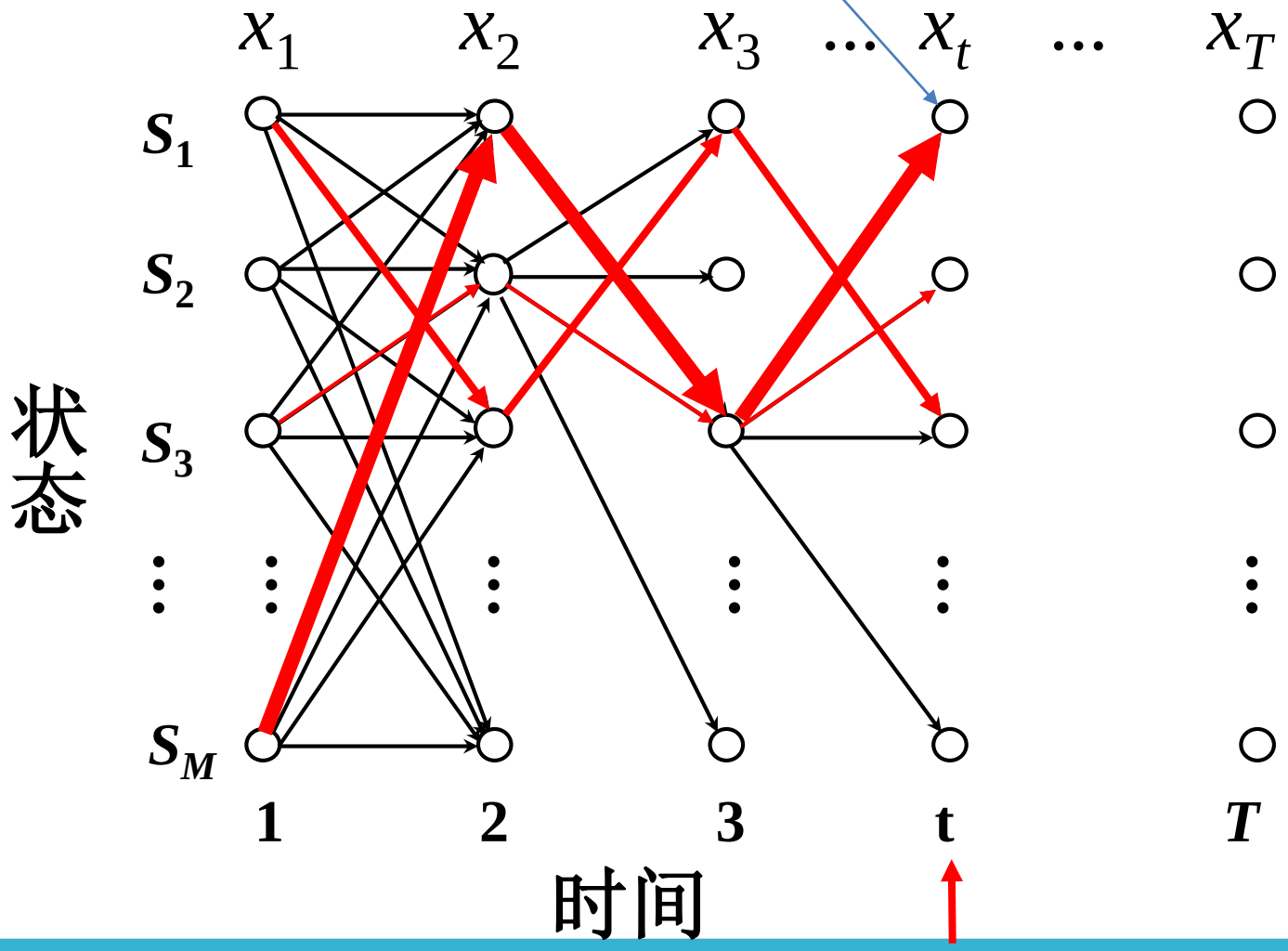
图解Viterbi搜索过程



$$\delta_t(i) = \max_{1\le j\le M}\{\delta_{t-1}(j)\cdot P(y_t(i)\mid y_{t-1}(j))\}\cdot P(x_t\mid y_t(i)), 2\le t\le T, 1\le j,i\le M$$

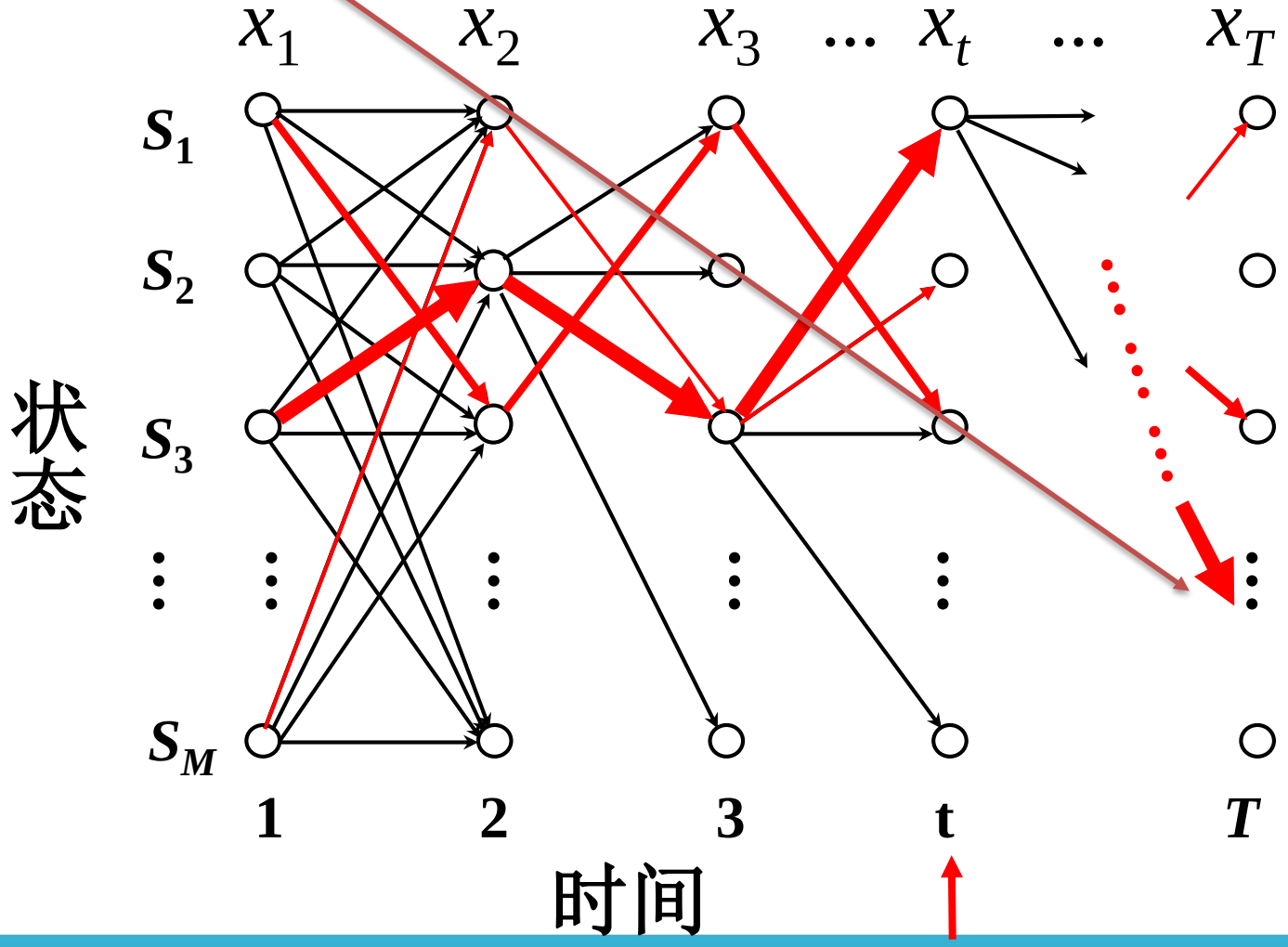
$$\delta_t(1) = \max_{1\le j\le M}\{\delta_{t-1}(j)\cdot P(y_t(1)\mid y_{t-1}(j))\}\cdot P(x_t\mid y_t(1))$$

图解Viterbi搜索过程



找到最大值 $\max_{1\le i\le M}\delta_T(i)$，然后通过回溯得到路径 $Y=\arg\max_{1\le i\le M}[\delta_T(i)]$

图解 Viterbi 搜索过程



- 输入观察序列X：南京市长江大桥
- 状态集合：

状态Y	Begin	Middle	End	Single
解释	词的开始字	词的中间字	词的结束字	单字成词
示例	南京的“南”	乒乓球的“乓”	南京的“京”	你

- 输出状态序列Y
- 给定HMM模型：{A,B, π}

字	南	京	市	长	江	大	桥
状态Begin	0.3	0.3	0.1	0.2	0.1	0.1	0.2
状态Middle	0.2	0.3	0.2	0.3	0.3	0.2	0.1
状态End	0.1	0.1	0.3	0.2	0.1	0.2	0.6
状态Single	0.2	0.2	0.2	0.1	0.4	0.2	0.1

南B 京M 市E 长B 江M 大M 桥E ⟹ 南京市， 长江大桥

条件随机场模型(Conditional Random Fields，CRF)

- 对汉字进行标注训练，不仅考虑了词语出现的频率，还考虑了上下文，具备较好的学习能力，对歧义词和未登录词的识别都具有良好的效果。

$$Y^* = \arg\max_Y P(Y \mid X;\lambda) = \frac{1}{Z(X)} \exp[\sum_j \lambda_j F_j(Y,X)]$$

$$Z(X) = \sum_Y \exp[\sum_j \lambda_j F_j(Y,X)]$$

$$F_j(Y,X) = \sum_{t=1}^{T} f_j(y_{t-1}, y_t, X, t)$$

- $F_j()$第j个特征函数，可以表示状态特征函数，或者状态转移函数
- λ_j第j个特征特征函数的权重
- $Z(X)$归一化因子

状态Y	Begin	Middle	End	Single
解释	词的开始字	词的中间字	词的结束字	单字成词
示例	南京的“南”	乒乓球的“乓”	南京的“京”	你

特征函数	释义
$f_1(x_t, y_t)$	状态特征函数：字x_t作为状态y_t出现的概率
$f_2(y_{t-1}, y_t)$	状态转移函数：4个状态，那么就是4*4的状态转移概率矩阵
$f_3(x_{t-1}, x_t, y_t)$ or $f_4(x_t, x_{t+1}, y_t)$	在状态y_t下上下文字转移概率(前后字)

$$P(y_t \mid x_t, \lambda) = \lambda_1 f_1(x_t, y_t) + \lambda_3 f_3(x_{t-1}, x_t, y_t) + \lambda_4 f_4(x_t, x_{t+1}, y_t) + \lambda_2 \max_{y'_{t-1} \in \{B,M,E,S\}} \{ f_2(y'_{t-1}, y_t) P(y'_{t-1} \mid x_{t-1}, \lambda) \}$$

字	南	京	市	长	江	大	桥
状态Begin							
状态Middle							
状态End							
状态Single							

$$P(y_t \mid x_t, \lambda) = \lambda_1 f_1(x_t, y_t) + \lambda_3 f_3(x_{t-1}, x_t, y_t) + \lambda_4 f_4(x_t, x_{t+1}, y_t) + \lambda_2 \max_{y'_{t-1} \in \{B,M,E,S\}} \{ f_2(y'_{t-1}, y_t) P(y'_{t-1} \mid x_{t-1}, \lambda) \}$$

字	南	京	市	长	江	大	桥
状态Begin	0.5						
状态Middle	0.2						
状态End	0.2						
状态Single	0.3						

$$P(B \mid 南, \lambda)$$
$$= \lambda_1 f_1(南, B) + \lambda_3 f_3(\text{null}, 南, B) + \lambda_4 f_4(南, 京, B) + \lambda_2 * 0$$
$$= \lambda_1 f_1(南, B) + \lambda_3 * 0 + \lambda_4 f_4(南, 京, B) + \lambda_2 * 0$$

$$P(M \mid 南, \lambda)$$
$$= \lambda_1 f_1(南, M) + \lambda_3 f_3(\text{null}, 南, M) + \lambda_4 f_4(南, 京, M) + \lambda_2 * 0$$
$$= \lambda_1 f_1(南, M) + \lambda_3 * 0 + \lambda_4 f_4(南, 京, M) + \lambda_2 * 0$$

$$P(y_t \mid x_t, \lambda) = \lambda_1 f_1(x_t, y_t) + \lambda_3 f_3(x_{t-1}, x_t, y_t) + \lambda_4 f_4(x_t, x_{t+1}, y_t) + \lambda_2 \max_{y'_{t-1} \in \{B,M,E,S\}} \{ f_2(y'_{t-1}, y_t) P(y'_{t-1} \mid x_{t-1}, \lambda) \}$$

字	南	京	市	长	江	大	桥
状态Begin	0.5	0.2					
状态Middle	0.2	0.4					
状态End	0.2	0.1					
状态Single	0.3	0.2					

$$P(B \mid 京, \lambda)$$

$$= \lambda_1 f_1(京, B) + \lambda_3 f_3(南, 京, B) + \lambda_4 f_4(京, 市, B)$$

$$+\lambda_2 \max_{y' \in \{B,M,E,S\}} \{ f_2(B,B)P(B \mid 南),\ f_2(M,B)P(M \mid 南),\ f_2(E,B)P(E \mid 南),\ f_2(S,B)P(S \mid 南) \}$$

$$= \lambda_1 f_1(京, B) + \lambda_3 f_3(南, 京, B) + \lambda_4 f_4(京, 市, B) +$$

$$\lambda_2 * \max_{y' \in \{B,M,E,S\}} \{ f_2(B,B)0.5,\ f_2(M,B)0.2,\ f_2(E,B)0.2,\ f_2(S,B)0.3 \}$$

$$P(M \mid 京, \lambda)$$

$$= \lambda_1 f_1(京, M) + \lambda_3 f_3(南, 京, M) + \lambda_4 f_4(京, 市, \mathrm{M})$$

$$+\lambda_2 \max_{y' \in \{B,M,E,S\}} \{ f_2(B,M)P(B \mid 南),\ f_2(M,M)P(M \mid 南),\ f_2(E,M)P(E \mid 南),\ f_2(S,M)P(S \mid 南) \}$$

$$P(y_t \mid x_t, \lambda) = \lambda_1 f_1(x_t, y_t) + \lambda_3 f_3(x_{t-1}, x_t, y_t) + \lambda_4 f_4(x_t, x_{t+1}, y_t) + \lambda_2 \max_{y'_{t-1} \in \{B,M,E,S\}} \{ f_2(y'_{t-1}, y_t) P(y'_{t-1} \mid x_{t-1}, \lambda) \}$$

字	南	京	市	长	江	大	桥
状态Begin	0.5	0.3	0.1	0.1	0.1	0.1	0.2
状态Middle	0.2	0.4	0.2	0.3	0.2	0.3	0.1
状态End	0.2	0.1	0.4	0.2	0.1	0.4	0.6
状态Single	0.3	0.2	0.3	0.1	0.2	0.2	0.1

南B 京M 市E 长B 江M 大M 桥E ⇒ 南京市， 长江大桥

目录

第一部分：如何建立词项词典？

- 文档解析(Parsing a document)
- 词条化 (Tokenization)
- 停用词 (Stop Words)
- 词项归一化 (Normalization)
- 词干还原 (Stemming)
- 词形归并 (Lemmatization)

第二部分：如何实现倒排记录表？

- 快速合并算法：带跳表的倒排记录表
- 包含位置信息的倒排记录表以及短语查询

停用词Stop Words

- 停用词表
  - 将词项按照文档集频率(collection frequency)，从高到底排列
  - 选取与文档意义不大，高频出现的词，比如，a, an, the, to, and, be …
- 停用词使用的趋势
  - 现代搜索引擎发展的趋势使用少量的停用词表
  - 现代IR系统更加关注利用语言的统计特性来处理常见词问题

消除停用词问题和可能的方法

- 优点：停用词消除可以减少term的个数
- 缺点：有时消除的停用词对检索是有意义的。
  - “的士”、“to be or not to be”
- 消除方法
  - 查表法
  - 基于文档频率

目录

第一部分：如何建立词项词典？

- 文档解析(Parsing a document)
- 词条化 (Tokenization)
- 停用词 (Stop Words)
- 词项归一化 (Normalization)
- 词干还原 (Stemming)
- 词形归并 (Lemmatization)

第二部分：如何实现倒排记录表？

- 快速合并算法：带跳表的倒排记录表
- 包含位置信息的倒排记录表以及短语查询

词项归一化Normalization

- 归一化：将文档和查询中的词条“归一化”成一致的形式
  - 例如希望USA和U.S.A.之间也能形成匹配
- 归一化的结果
  - 在IR系统的词项词典中，形成多个近似词项的一个等价类
- 隐式的建立等价类
  - 例如将USA和U.S.A.映射为USA
  - 例如将anti-discrimination和antidiscrimination映射为antidiscrimination

词项归一化：不同语言之间的区别

- 重音符号
  - e.g.：法语中résumé vs. resume
- 变音符号
  - e.g.：德语中Tuebingen vs. Tübingen. (其实它们应该是等价的)
- 最重要的标准
  - 最重要的问题不是规范或者语言学的问题，而是用户将会如何根据这些词来构造查询？
- 即使在一些语言中，有的词有了标准的读音，但是用户有自己的读音/拼写方式
  - e.g.：Tuebingen, Tübingen, Tubingen

词项归一化：不同语言之间的区别

- 其他
  - 中文中日期的表示7月30日 vs. 英文中7/30
  - 日语中使用的假名汉字 vs. 中文中的汉字
- 词条化和归一化
  - 二者都依赖于不同的语言种类，因此，在整个索引建立过程中要综合考虑
  - e.g.： *Morgen will ich in MIT …*

Is this German "mit"?

德语***Morgen will ich in MIT*** 的意思是“我明天在MIT”，而德语中的“MIT”其实是“与”的意思

词项归一化：大小写转换

- 一般策略
  - 将所有字母转换为小写
  - 绝大多数情况下，用户在构造查询时都忽略首字母的大写
  - 一些专有名词除外
    - e.g.：General Motors
    - Fed vs. fed
    - SAIL vs. sail
- Google的例子
  - 输入查询词C.A.T.，首页是关于猫的网站，而不是卡特彼勒公司 (Caterpillar Inc.) (2005年的时候)

词项归一化

- 词项归一化的策略：建立同义词扩展表。

- 例子：

查询:	检索:
window	window, windows
windows	Windows, windows, window
Windows	Windows

扩展词表和soundex算法

- 如何处理同义词和同音词？
  - e.g.：手工建立同义词词表
    - car = automobile, color = colour
  - ①为每个查询维护一张包含多个词的查询扩展词表
    - 例如：查询automobile的同时，也查询car
  - ②在建立索引时就对词进行扩展
    - 例如：对于包含automobile的文档，同时也使用car来索引
- 如何处理拼写错误？
  - 其中的一种处理方法，就是根据发音相同来进行词项扩展
  - 后续章节中有讨论

目录

第一部分：如何建立词项词典？

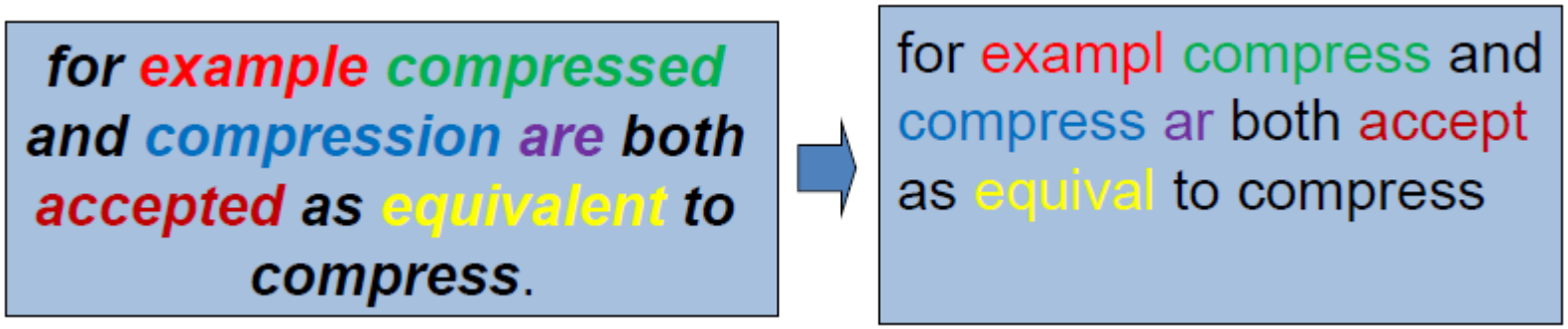
- 文档解析 (Parsing a document)
- 词条化 (Tokenization)
- 停用词 (Stop Words)
- 词项归一化 (Normalization)
- 词干还原 (Stemming)
- 词形归并 (Lemmatization)

第二部分：如何实现倒排记录表？

- 快速合并算法：带跳表的倒排记录表
- 包含位置信息的倒排记录表以及短语查询

词干还原(Stemming)

- 通常指很粗略的去除单词两端词缀的启发式过程。
  - e.g., automate(s), automatic, automation → automat



- 词干还原能够提高召回率，但是会降低准确率
  - e.g.: operative ⇒ oper
  - 词干还原对于芬兰语，西班牙语，德语，法语都有明显的作用，其中对芬兰语的提高达到30%(以MAP平均准确率来计算)。

中文重叠词还原—可视为“词干还原”

形容词(AB)	ABAB式	AABB式	A里AB式
高兴	高兴高兴	高高兴兴	
明白	明白明白	明明白白	
热闹	热闹热闹	热热闹闹	
潇洒	潇洒潇洒	潇潇洒洒	
糊涂		糊糊涂涂	糊里糊涂
流气			流里流气
粘乎	粘乎粘乎	粘粘乎乎	

形容词A	ABB式	ABCD式
黑	黑压压	黑不溜秋
白	白花花	白不呲咧
红	红彤彤	
亮	亮晶晶	

Porter算法

- 英文处理中最常用的词干还原算法。
  - 经过实践证明是高效性的算法。

- 算法包括5个按照顺序执行的词项约简步骤
  - 每个步骤都是按照一定顺序执行的
  - 每个步骤中包含了选择不同的规则的约定
  - 比如，从规则组中选择作用时词缀最长的那条规则

Porter算法中的典型规则

- sses→ss
- ies→i
- ational→ate
- caresses→caress
- ponies→poini
- national→nate

- 要考虑规则的“权重”
- (前面的字母数>1) EMENT →
  - replacement → replac
  - cement → cement

词形归并(Lemmatization)

- 利用词汇表和词形分析来减少屈折变化的形式，将其转变为基本形式。
  - e.g.
    - am, are, is → be
    - car, cars, car's, cars' →car
    - the boy's cars are different colors →the boy car be different color
- 词形归并可以减少词项词典中的词项数量

词干还原（Stemming)和词形归并（Lemmatization)

- 代表意义不同。
- 前者：通常指很粗略的去除单词两端词缀的启发式过程。
- 后者：通常指利用词汇表和词形分析来去除屈折词缀，从而返回词的原形或词典中的词的过程。
- 假如给定词条 saw，
  - 词干还原过程可能仅返回 s，
  - 而词形归并过程将返回see或者saw，
  - 具体返回哪个词取决于当前上下文中saw是动词还是名词。
- 两个过程的区别还在于
  - 词干还原在一般情况下会将多个派生相关词合并在一起，
  - 而词形归并通常只将同一词元的不同屈折形式进行合并。

语言的特殊性

- 词干还原和词形归并，都体现了不同语言之间的差异性，包括
  - 不同语言之间的差异
  - 特殊专业语言与一般语言的差异
- 词干还原或者词形归并往往通过在索引过程中增加插件程序的方式来实现
  - 商业软件
  - 开源软件

目录

第一部分：如何建立词项词典？

- 文档解析(Parsing a document)
- 词条化 (Tokenization)
- 停用词 (Stop Words)
- 词项归一化 (Normalization)
- 词干还原 (Stemming)
- 词形归并 (Lemmatization)

第二部分：如何实现倒排记录表？

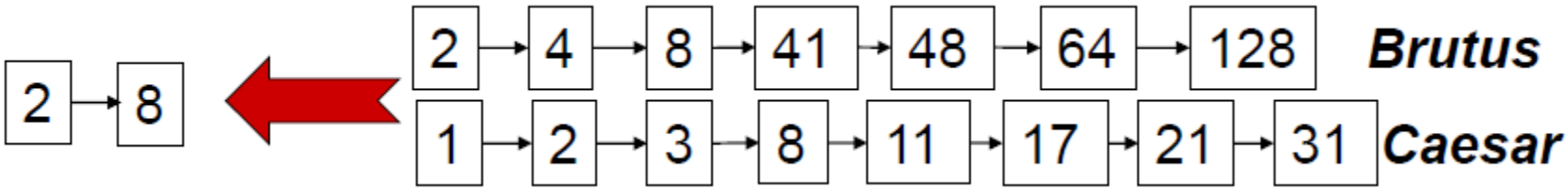
- 快速合并算法：带跳表的倒排记录表
- 包含位置信息的倒排记录表以及短语查询

单选题 1分

设置

合并算法(Postings Merges)回顾

通过在二个倒排表之间同时移动指针来实现合并，此时的操作与线性表的总数成线性关系。



如果倒排表的长度分别是*m*和*n*，那么合并算法需要操作（）次。

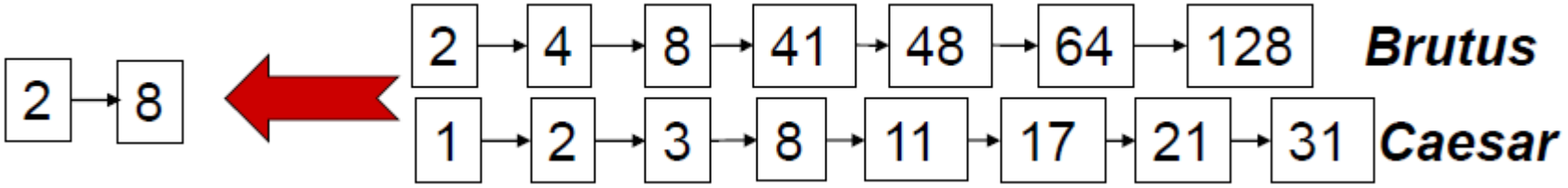
A O(m+n)

B O(mn)

提交

合并算法(Postings Merges)回顾

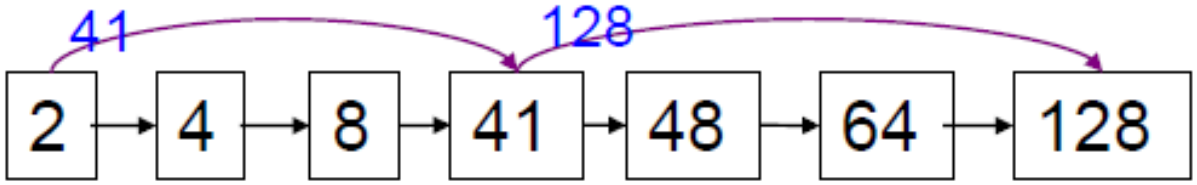
- 通过在二个倒排表之间同时移动指针来实现合并，此时的操作与线性表的总数成线性关系。

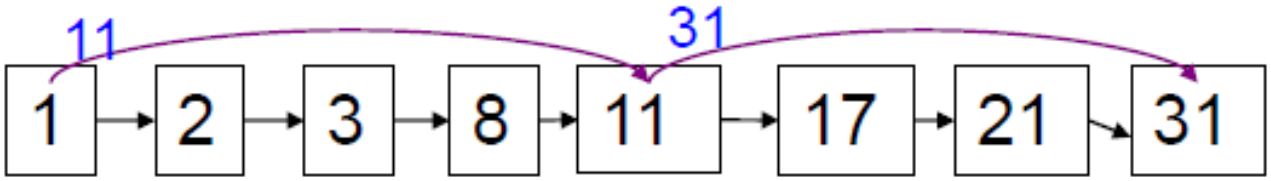


- 如果倒排表的长度分别是*m*和*n*，那么合并算法需要操作O(*m*+*n*)次。
- 能否做的更好？

基于跳表(Skip List)的倒排记录表快速合并算法

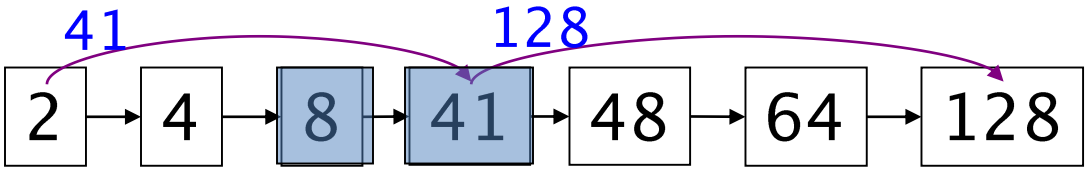
- 跳表指针能够跳过那些不可能出现在检索结果中的记录项。
- 构建跳表的二个主要问题
  - 如何利用跳表指针进行快速合并？
  - 在什么位置设置跳表指针？

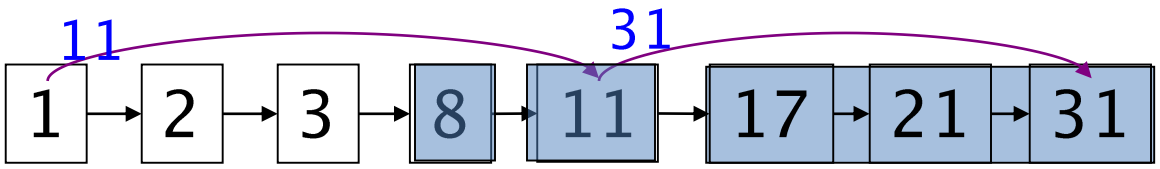




带有跳表指针的查询处理过程

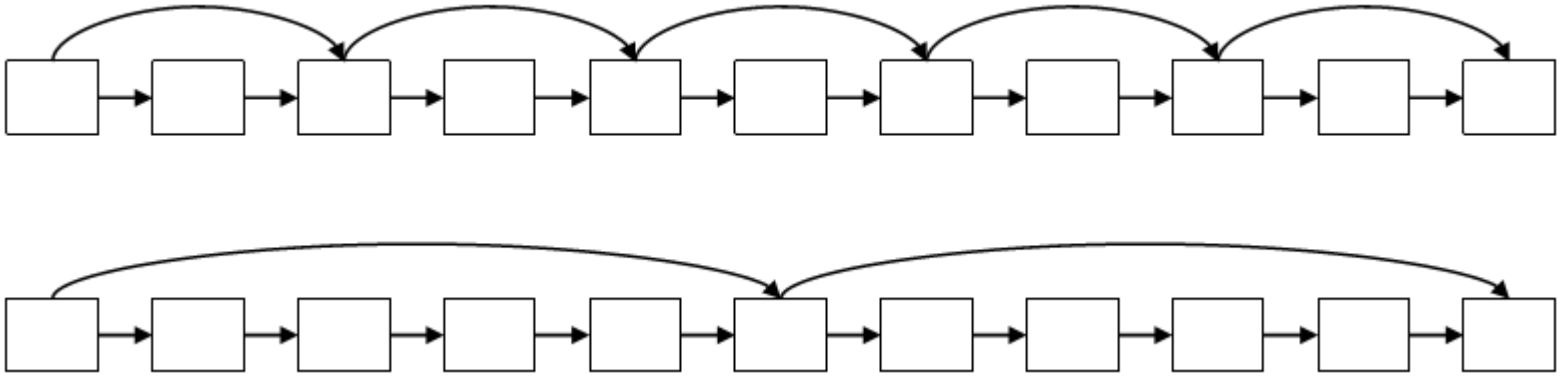
- 1.假定进行遍历一直发现共同记录8，将8放入结果表中之后，继续移动二个表的指针。
- 2.假定第一个表指针移到41，第二个表的指针移到11。
- 3.由于11 < 41，因此，上面的指针不需要继续移动，只需移动下面的指针，跳到31。(当然如果31>41，那么就要倒回来)
- 4.这样就跳过了17,21。





在什么位置设置跳表指针？

- 策略
  - 设置较多的指针 → 较短的步长 ➔更多的跳跃机会
    - 更多的指针比较次数和更多的存储空间
  - 设置较少的指针→较长的步长 ➔ 较少的连续跳跃
    - 较少的指针比较次数和较少的存储空间



设置跳表指针

- 放置跳表指针的一个简单的启发式策略是：
  如果倒排表的长度是L，那么在每个$\sqrt{L}$处均匀放置跳表指针
- 该策略没有考虑到查询词项的分布
- 如果索引相对固定的话，建立有效的跳表指针比较容易，如果索引需要经常的更新，建立跳表指针就相对困难。
- 硬件参数对索引构建有一定的影响
  - CPU速度
  - 磁盘访问速度
- 注意：跳表指针只对AND类型的查询有用，对OR类型的查询不起作用。

目录

第一部分：如何建立词项词典？

- 文档解析(Parsing a document)
- 词条化 (Tokenization)
- 停用词 (Stop Words)
- 词项归一化 (Normalization)
- 词干还原 (Stemming)
- 词形归并 (Lemmatization)

第二部分：如何实现倒排记录表？

- 快速合并算法：带跳表的倒排记录表
- 包含位置信息的倒排记录表以及短语查询
  - 方法1：二元词索引
  - 方法2：位置信息索引

短语查询(Phrase Query)

- 用户希望将类似“stanford university”的查询中的二个词看成是一个整体。
- 类似“I want to university at stanford”这样的文档是不会被匹配的。
  - 大部分的搜索引擎都支持双引号的短语查询，这种语法很容易理解并被用户成功使用。
  - 有很多查询在输入时没有加双引号，其实都是隐式的短语查询(如人名)。
  - 要支持短语查询，只记录<term：docs> 这样的条目是不能满足用户需要的。

第一种方法：二元词索引(Biword indexes)

- 将文档中每个连续词对看成一个短语
- 例如，文本“Friends, Romans, Countrymen”将生成如下的二元连续词对
  - friends romans
  - romans countrymen
- 其中的每一个二元词对都将作为词典中的词项
- 经过上述的处理，此时可以处理二个词构成的短语查询

扩展的二元词索引

- 名词和名词短语构成的查询具有相当特殊的地位。
  - 首先对文本进行词条化，然后进行**词性标注**
  - 把每个词项分为名词(N)、虚词(X)，冠词和介词和其他词。
  - 将形式为N*XN非词项序列看成一个扩展的二元词
  - 每个这样的扩展的二元词对应一个词项
- 例如：catcher in the rye

  N X X N
- 利用这样的扩展二元词索引处理查询，
  - 将查询拆分成N和X
  - 将查询划分成扩展的二元词
  - 最后在索引中进行查找

第二种方法：位置信息索引(Positional indexes)

- 在这种索引中，对每个词项，采取以下方式存储倒排表记录

〈词项，词项频率；
文档1：位置1，位置2……；
文档2：位置1，位置2……；
……〉

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

- 对于短语查询，仍然采用合并算法，查找符合的文档不只是简单的判断二个词是否出现在同一文档中，还需要检查它们出现的位置情况

短语查询过程

- 例子

查询词：“to be or not to be”

倒排表：

- *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; . . .

1. 考虑to和be的倒排表的合并，查找同时包含to和be的文档
2. 检查表中，看看是否某个be的前面的一个位置上正好出现to

邻近查询(Proximity queries)

- Employ me/3 place，表示从左边或右边相距在*k*=3个词之内
- 显然，位置索引能够用于邻近搜索，而二元词搜索则不能

位置信息索引的讨论

- 采用位置索引会大大增加倒排记录表的存储空间，即使采用后面讨论的压缩方法也无济于事。
- 由于用户期望能够进行短语查询和邻近查询，所以还是得采取这种索引方式。

经验法则(English- like)

- 位置索引大概是非位置索引大小的2~4倍
- 位置索引的大小大约是原始文档的30%~50%

混合索引机制

- 二元词索引和位置索引二种策略可以进行有效的合并
  - 对于高频查询词可以采用二元词索引，例如“*Michael Jackson*”，
- Williams等人(2004)评估了更复杂的混合索引机制，(引入后续词索引方法)。
  - 对于一个典型的web短语混合查询，其完成时间大概是只使用位置索引的1/4
  - 比只使用位置索引增加26%的空间