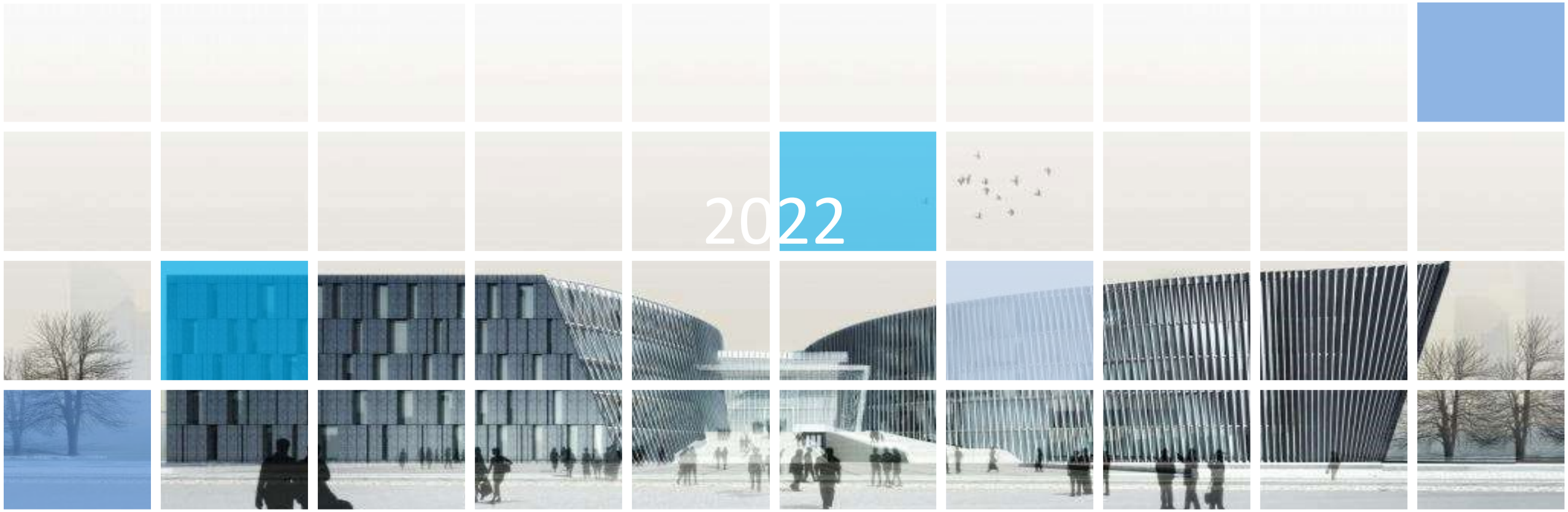


南京大学信息管理学院

信息检索

邓三鸿

njuir@sina.com

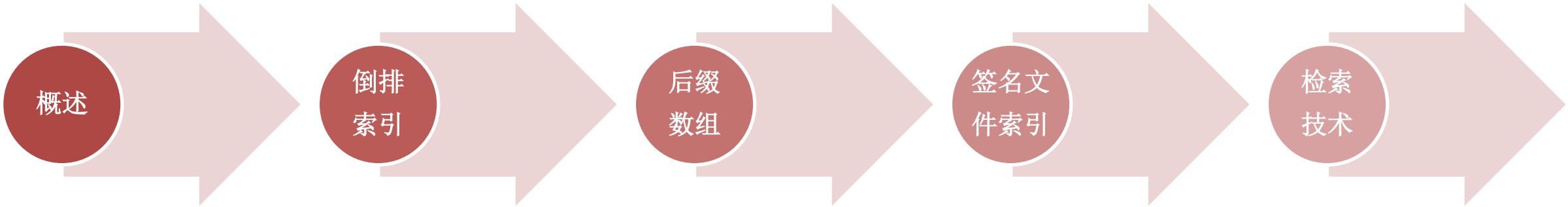
07

PART Seven

文本索引和搜索技术

Index & Retrieval Tech.

索引技术的内容



为什么要索引？

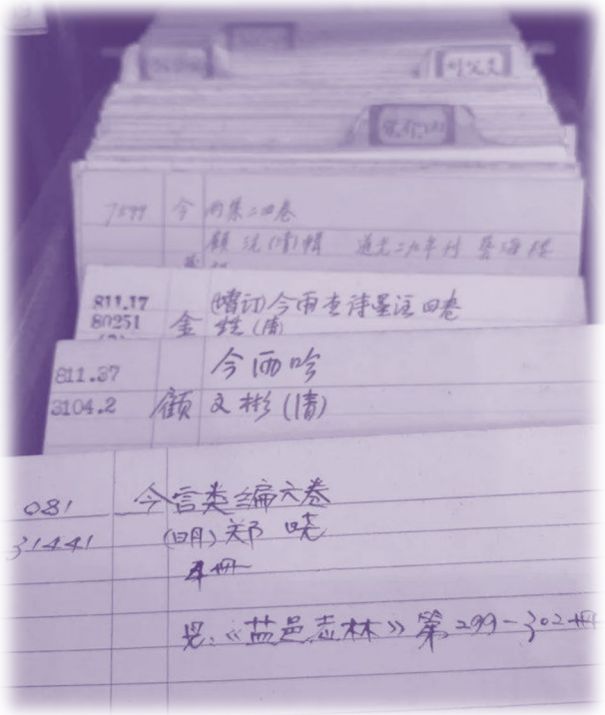
我的肉心其实是崩溃的

索引与标引？

- Index
- 标引
- **索引：IR中，对照或引导标引信息的排列表**
  - 主题索引
    - 关键词索引、单元词索引、标题词索引、叙词索引……
  - 类号索引
  - 引文索引（SCI、EI、CSSCI……）

索引表现形式

- 将标引的结果（主题、类号）**按照一定规律排列**的处理技术
- 任何检索工具都应该由**二次文献部分**和**检索标识**组成



检索工具	二次文献形式	主题表现形式
卡片式	文献卡片	导卡
书本式	文献条目	主题索引
机读顺排文档	条目字段	主题词字段
机读倒排文档	条目区	主题词区

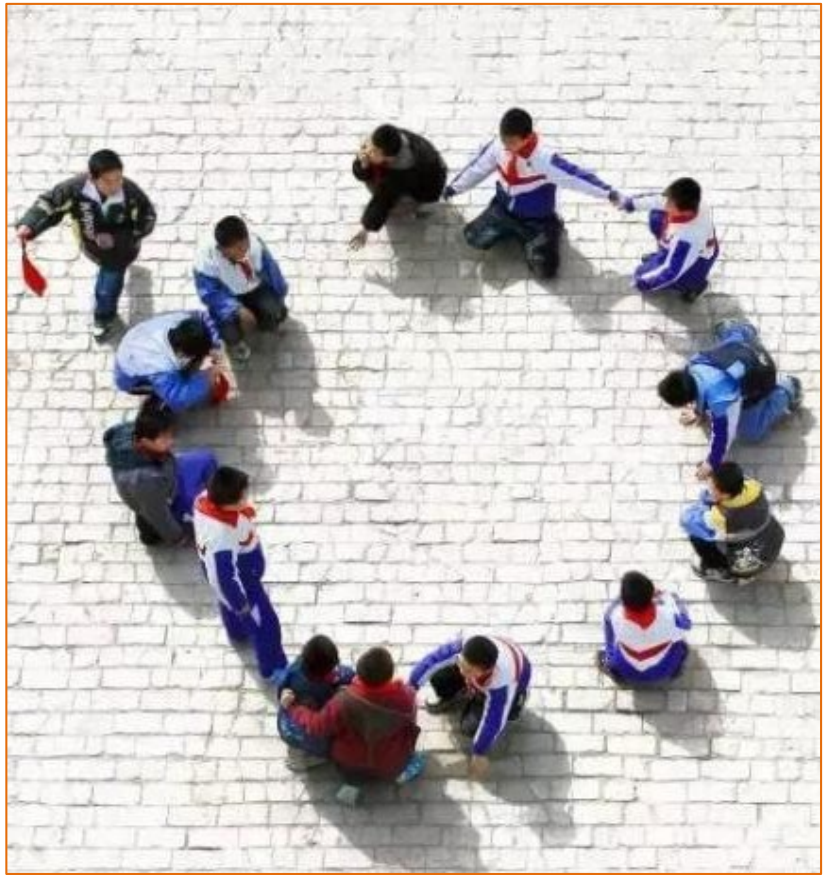
索引的（存储）价值

- 所谓建立索引，是指将待搜索的信息进行一定的**分析**，并将分析的结果按照**一定的组织方式存储**起来，通常是存储在文件中.
- 存储了分析结果的文件的集合就是所谓的**索引**.
- 准确定义：**索引**（Index）是一种**数据结构**，其将关键词与包含该关键词的文档（或关键词在文档中的位置）建立了一种映射关系，以**加快检索的速度**。

索引的存储方法

顺排文档索引

- 思想
  - ✓ 将文档中的每一条记录**依次去匹配**用户的检索提问集合，文档处理完毕后，将各提问的命中结果归并分发给有关用户。
- 定义
  - ✓ 用文档中记录一条一条去匹配提问的，是顺序对文档记录检索的方法
- 关键技术
  - ✓ 采用列表处理方法将提问逻辑式（检索式）变换成等价的提问展开式，按提问展开表的内容对顺排文档的每篇文献进行检索
- 查询方法
  - ✓ ***表展开法***、*逻辑树法等*



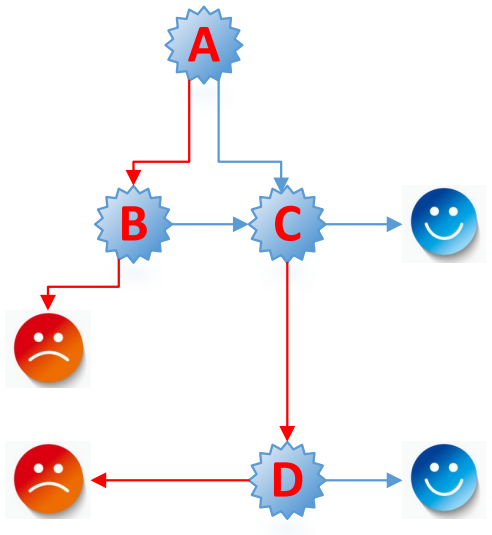
查询的表展开法

1968年菊池敏典提出，其主要思想是采用列表处理方法将逻辑提问式即检索式变换为等价的**提问展开表**，然后按提问展开表的内容对文档中的每一条记录进行检索。

将经典**布尔逻辑检索**的逻辑提问表达式转换为逻辑检索表，每个检索词的检索组配关系要求能够用表进行精确映射，检索的记录是否最终命中检索需求要能准确反映出来。

(A+B) * (C+D) 的展开检索基础表

地址	检索词	条件满足指向	条件非满足指向
1	A	3	2
2	B	3	落选
3	C	命中	4
4	D	命中	落选



特点

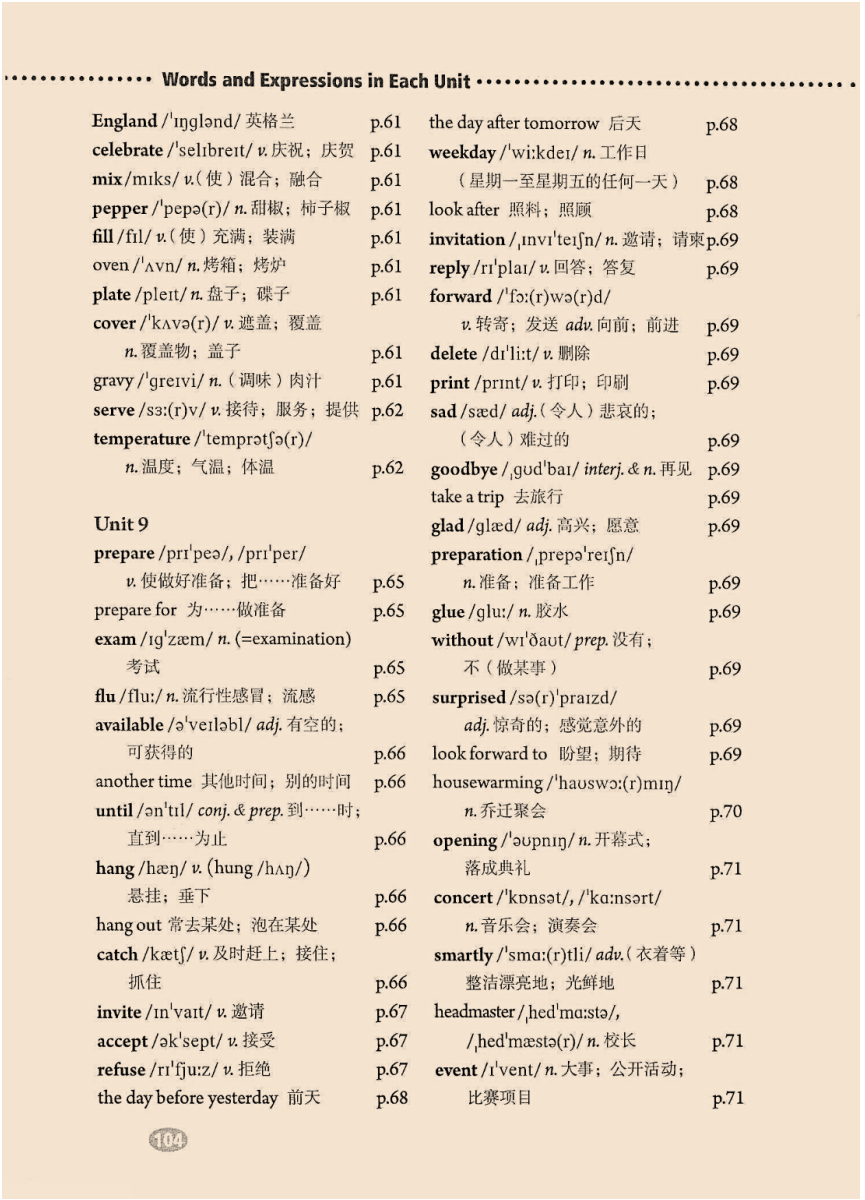
- **优点**
  - 凡是可不查阅的文献属性一定不查
  - 凡是可不再查阅的检索词一定不再查
  - 节省机器的查比时间，早期得到广泛应用
- **缺点**
  - 其效率在某种程度上依赖原提问式的书写.
  - 在实际的定额批式检索中，很多提问中往往含有很多相同的检索词，由于每个提问都要与主文档进行匹配处理，因此一批提问中这些相同的检索词和同一文献要重复匹配多次

倒排文档

（Inverted File）

也称**倒排索引**，索引对象是文档或文档集合中的**单词**等，用来存储这些单词在一个文档或者一组文档中的**存储位置**，是对文档或文档集合的一种**最常用的索引机制**。

如:有些书往往在最后提供的索引(单词—页码列表表)就可以看成是一种倒排索引，即通过一些关键词，在全书中检索出与之相关的部分。



Words and Expressions in Each Unit

England /ˈɪŋgləndˌ/ 英格兰 p.61
celebrate /ˈselɪbreɪt/ *v.* 庆祝；庆贺 p.61
mix /mɪks/ *v.*（使）混合；融合 p.61
pepper /ˈpepə(r)/ *n.* 甜椒；柿子椒 p.61
fill /fɪl/ *v.*（使）充满；装满 p.61
oven /ˈʌvn/ *n.* 烤箱；烤炉 p.61
plate /pleɪt/ *n.* 盘子；碟子 p.61
cover /ˈkʌvə(r)/ *v.* 遮盖；覆盖 *n.* 覆盖物；盖子 p.61
gravy /ˈgreɪvi/ *n.*（调味）肉汁 p.61
serve /sɜː(r)v/ *v.* 接待；服务；提供 p.62
temperature /ˈtemprətʃə(r)/ *n.* 温度；气温；体温 p.62

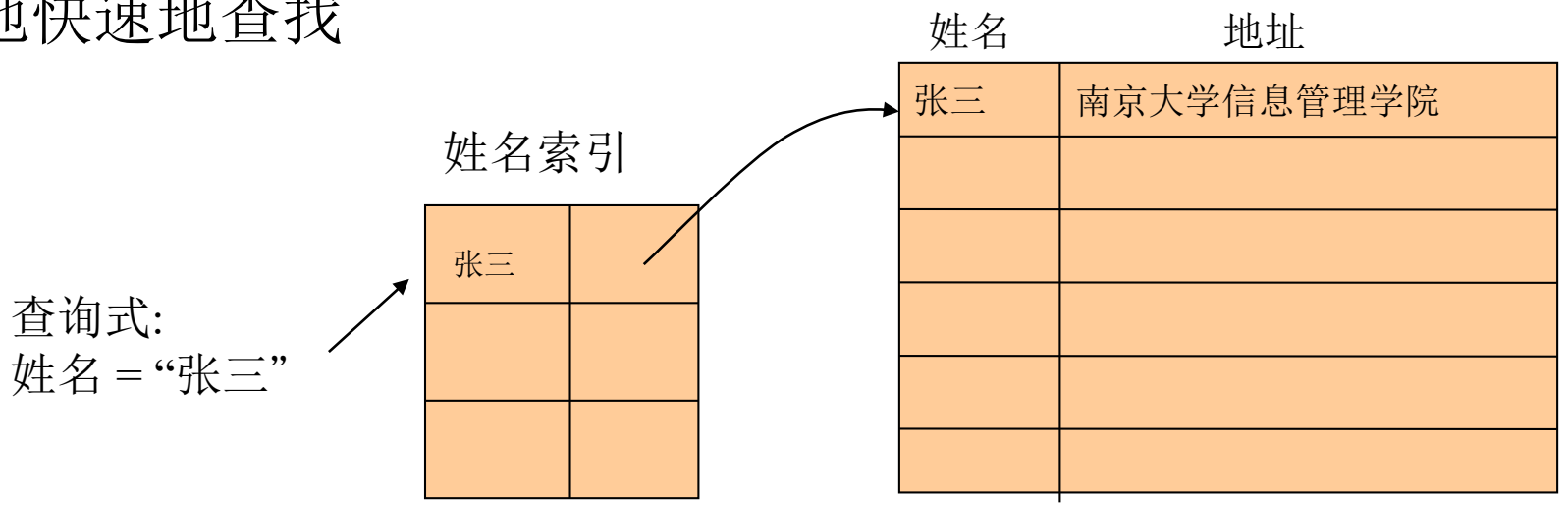
Unit 9

prepare /prɪˈpeə/, /prɪˈper/ *v.* 使做好准备；把……准备好 p.65
prepare for 为……做准备 p.65
exam /ɪgˈzæm/ *n.* (=examination) 考试 p.65
flu /fluː/ *n.* 流行性感冒；流感 p.65
available /əˈveɪləbl/ *adj.* 有空的；可获得的 p.66
another time 其他时间；别的时间 p.66
until /ənˈtɪl/ *conj.* & *prep.* 到……时；直到……为止 p.66
hang /hæŋ/ *v.* (hung /hʌŋ/) 悬挂；垂下 p.66
hang out 常去某处；泡在某处 p.66
catch /kætʃ/ *v.* 及时赶上；接住；抓住 p.66
invite /ɪnˈvaɪt/ *v.* 邀请 p.67
accept /əkˈsept/ *v.* 接受 p.67
refuse /rɪˈfjuːz/ *v.* 拒绝 p.67
the day before yesterday 前天 p.68
the day after tomorrow 后天 p.68
weekday /ˈwiːkdeɪ/ *n.* 工作日（星期一至星期五的任何一天） p.68
look after 照料；照顾 p.68
invitation /ˌɪnvɪˈteɪʃn/ *n.* 邀请；请柬 p.69
reply /rɪˈplaɪ/ *v.* 回答；答复 p.69
forward /ˈfɔː(r)wə(r)d/ *v.* 转寄；发送 *adv.* 向前；前进 p.69
delete /dɪˈliːt/ *v.* 删除 p.69
print /prɪnt/ *v.* 打印；印刷 p.69
sad /sæd/ *adj.*（令人）悲哀的；（令人）难过的 p.69
goodbye /ˌgʊdˈbaɪ/ *interj.* & *n.* 再见 p.69
take a trip 去旅行 p.69
glad /glæd/ *adj.* 高兴；愿意 p.69
preparation /ˌprepəˈreɪʃn/ *n.* 准备；准备工作 p.69
glue /gluː/ *n.* 胶水 p.69
without /wɪˈðaʊt/ *prep.* 没有；不（做某事） p.69
surprised /sə(r)ˈpraɪzd/ *adj.* 惊奇的；感觉意外的 p.69
look forward to 盼望；期待 p.69
housewarming /ˈhaʊswɔː(r)mɪŋ/ *n.* 乔迁聚会 p.70
opening /ˈəʊpnɪŋ/ *n.* 开幕式；落成典礼 p.71
concert /ˈkɒnsət/, /ˈkɑːnsərt/ *n.* 音乐会；演奏会 p.71
smartly /ˈsmɑː(r)tli/ *adv.*（衣着等）整洁漂亮地；光鲜地 p.71
headmaster /ˌhedˈmɑːstə/, /ˌhedˈmæstə(r)/ *n.* 校长 p.71
event /ɪˈvent/ *n.* 大事；公开活动；比赛项目 p.71

104

在关系数据库上建索引

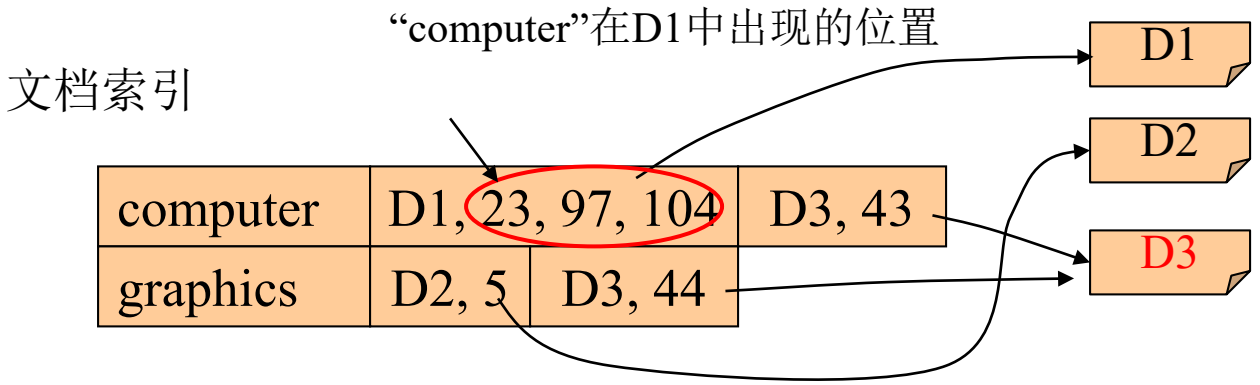
- 这种想法也被应用于数据库技术中，即对数据库中需要经常进行检索的域建立索引结构，进行快速的查询。
- 索引结构: *hashing, B+-tree*
- 可以索引全部记录，在全部记录上进行搜索
- 精确地快速地查找



倒排文档的组成

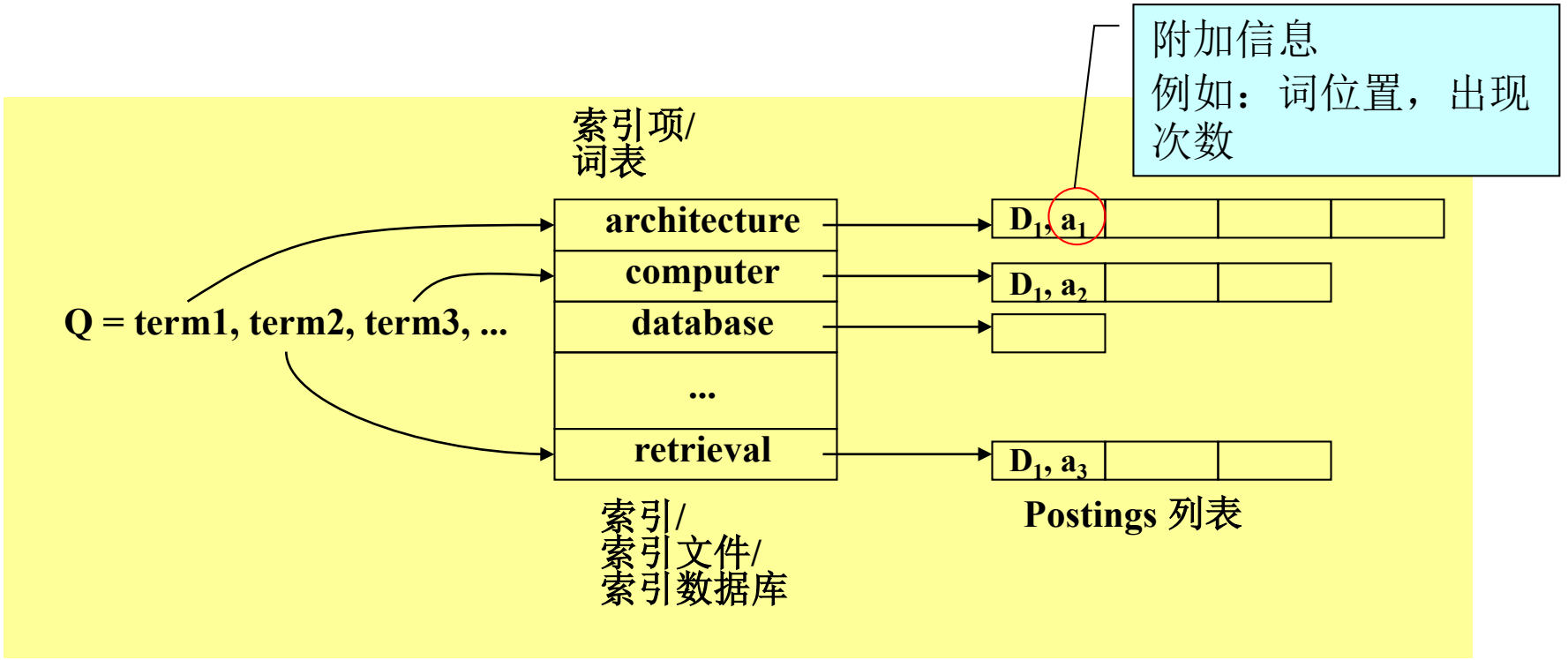
- 倒排文档一般由两部分组成：词汇表（vocabulary）和记录表（posting list）
- 词汇表是文本或文本集合中所包含的所有不同单词（索引项）的集合。
- 对于词汇表中的每一个单词，其在文本中出现的位置或者其出现的文本编号构成一个列表，所有这些列表的集合就称为记录表。

对文档进行索引



- 可以进行部分匹配: ’%comput% ’
- 可以进行短语搜索:查找包含 “computer graphics”的文档

一般的倒排索引



- 索引文件可以用任何文件结构来实现
- 索引文件中的词项是文档集合中的词表

例子

文本

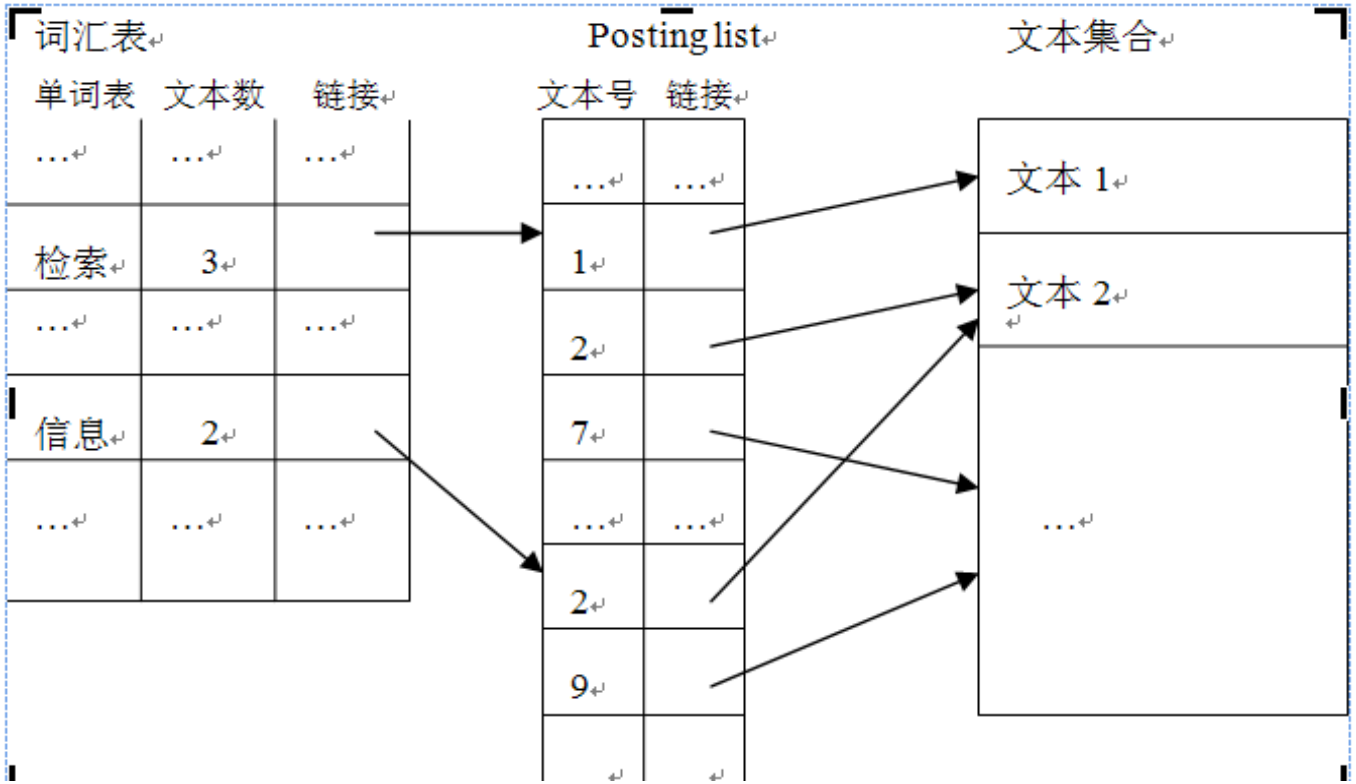
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
这	是	一本	关于	信息	检索	的	教材	。	介绍	了	检索	的	基本	技术	。	…

倒排文件

词汇表	记录表
技术	15, …
教材	8, …
检索	6, 12, …
信息	5, …
…	…

以文本为记录表

记录表既可以存储文本中单词的编号位置，也可以指向单词首字母的字符位置，还可以是其所在的文本编号，下图是一个以文本为记录表的情况



倒排文档的使用

- 词汇表检索
  - 将出现在查询中的单词分离出来，并在词汇表中进行检索；
- 记录表检索
  - 检索出所有找到的单词对应的记录表；
- 记录表操作
  - 对检索出的记录表进行处理，实现短语查询、相邻查询或布尔查询等。

提问式的逆波兰展开

- 逆波兰
  - 1929年波兰的逻辑学家卢卡西维兹提出将运算符放在运算项后面的逻辑表达式，又称“逆波兰表达式”（Reverse Polish Notation，RPN，或逆波兰记法），也叫后缀表达式（将运算符写在操作数之后）
  - 用于数据结构等多个场合
- 福岛算法
  - 例如：逻辑提问式 A*(B+C)+D （中缀表达式）
    逆波兰表达式： ABC+*D+ （后缀表达式）
    波兰表达式：+*A+BCD （前缀表达式）

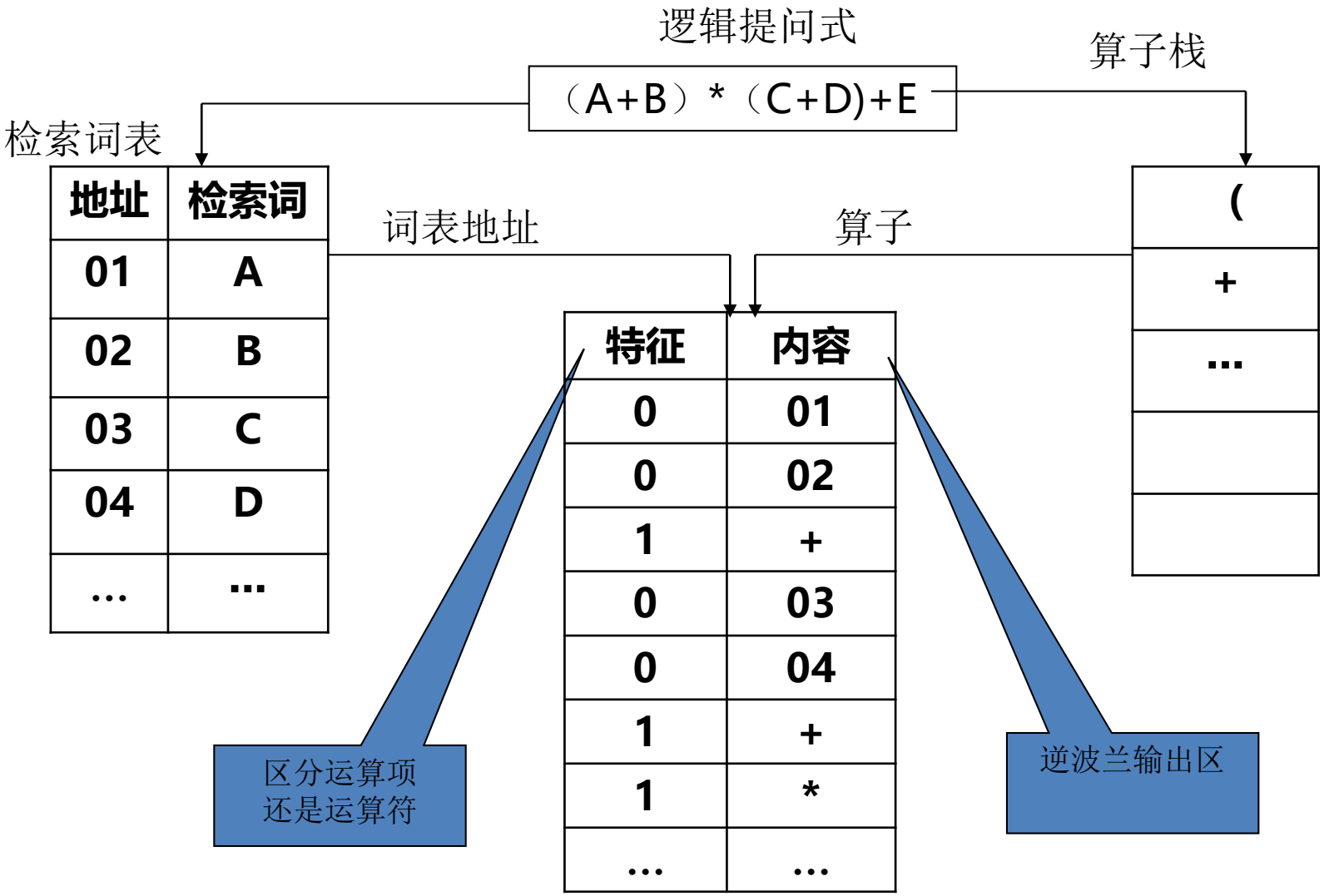
Jan Łukasiewicz（1878-1956）

- **福岛（Fukujima）算法首先要进行提问式的转换。**

逆波兰表达式

- 中缀表达式生成的逆波兰表达式是唯一的
- 注意：
  - a+(b-c)*d 和 (b-c)*d+a 和 a+d*(b-c) 的值是完全一样的。但是，中缀形式不同，产生的逆波兰式必然是不同的。
  - a+(b-c)*d ： abc-d*+
  - (b-c)*d+a ： bc-d*a+
  - a+d*(b-c) ： adbc-*+

逆波兰法处理示意图



三个存储区

- 算子栈
- 检索词表
- 逆波兰输出区

检索过程示例

- 以提问式（A+B）*（C+D）+E为例，说明其工作区占用情况，以及由逆波兰表示式转换为一组检索指令的过程。

工作区W

1	A结果文献号码集合 1）	C结果文献号码集合 4）	(A+B)*(C+D)结果文献号码集合 7）	…
2	B结果文献号码集合 2）	D结果文献号码集合 5）	E结果文献号码集合 8）	
3	A和B“或”结果文献号码集合 3）		(A+B)*(C+D)和E“或”结果文献号码集合 9）	
4	C和D“或”结果文献号码集合 6）			
…	…			
n=7	(A+B)*(C+D)+E.结果文献号码集合 10）	转储		

倒排索引的特点

- 快速索引（长query需要更多时间）；
- 灵活性: 不同类型的信息都可以存储在记录表中；
- 如果存储了足够多的信息，则可以支持复杂的检索操作；
- 存储开销较大；
- 更新、插入和删除都需要很高的维护开销，倒排索引相对静态的环境（很少插入和更新）中使用比较好。

后缀数组

- 在倒排文档中，文本被看作是由单词组成的序列➔限制了倒排文件的应用
  - 情况1：词组查询在使用倒排文件查询时就比较困难，因为不但需要记录每个单词在文档中的位置，还要在合并时判断其是否相邻并构成词组；
  - 情况2：在某些应用中，如基因数据库，不存在词的概念。
- 解决方案：使用**后缀数组（suffix array）**
- 在后缀数组中，可以将文本看作是一个长的字符串，文本中的每个位置都被认为是文本的一个后缀，即一个从当前文本位置到文本末尾的字符串。
- 索引的位置可以是每个字符的位置、单词的位置或者汉字的位置等。
- **后缀数组**就是对文本中的所有后缀按照词典序存放每个后缀对应的起始位置的一个列表

后缀数组的构造

原始文本，按字的顺序位置索引

0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40	42	44	46	48	…
这	是	一	本	关	于	信	息	检	索	的	教	材	。	介	绍	了	检	索	的	基	本	技	术	。	…

首先截取每个后缀的前n个字节，作为类似倒排文件中的“词汇表”(此处n=4，2汉字)

0	2	12	16	22	34	44	…
这是…	是一…	信息…	检索…	教材…	检索…	技术…	…

文本中的部分后缀，按位置索引

44	16	34	22	2	12	0	…
技术…	检索…	检索…	教材…	是一…	信息…	这是…	…

相同的部分后缀，按字典序索引

后缀数组的使用

- 在使用后缀数组进行检索的时候，将每个查询同样截取前*n*个字节，并于索引中进行查找
  - 如果没有找到，则表明不包含所需查询
  - 如果查找成功，则需要在相应的文本位置上，进行进一步的字符串比较，以确定文本中是否包含查询
- 实例
  - 查找“信息检索”，首先截取4个字节“信息”，并于后缀数组中查找到了位置12，接着在原文中的12号位置，找到“信息检索”字符串，则查找成功；
  - 查找“信息过滤”，则原文中的12号位置不能找到“信息过滤”字符串，则查找失败；
  - 更一般的例子，如果输入的查询为“数据库”，在索引结构中不能找到“数据”，则查找直接失败返回；

44	16	34	22	2	12	0	…
技术…	检索…	检索…	教材…	是一…	信息…	这是…	…

后缀数组的分析

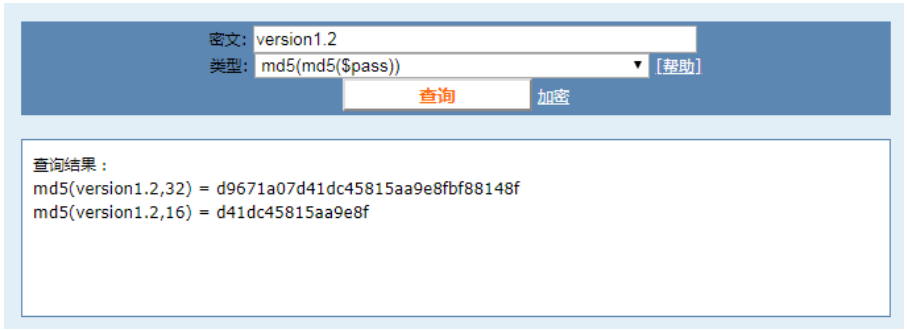
- 对于需要大数据量的检索问题，后缀数组并不适用
- 因为构造出的后缀数组需要占用大量的空间，通常是原文本的1.7倍
- 和倒排文档相比，后缀数组里面储存了较多的重复信息
- 同时，由于每个后缀位置的编号不能存储相对位置的变化，所以很难被压缩，需要较多的存储空间；
- 后缀数组的另外一个缺点是不容易对检索结果进行排序，因为计算查询单词的词频比较耗时；
- **实际上,后缀数组的大部分功能可以通过倒排文档来实现!**
  - 例如可以倒排索引文本中的二字串或者三字串，从而提高召回率

签名文件

- **签名文件（signature file）**是基于散列（Hash）技术的面向单词的索引结构
- 索引占用的空间大约为原始文档集的30~40%
- 在检索时需要顺序比较，适用于小规模的文本
- 在大多数应用中，其性能不如倒排文件

关于Hash

- **散列/哈希**
  - ✓ 任意长度的数据映射到有限长度的域上
- **要求**
  - ✓ 抗冲突能力
  - ✓ 抗篡改能力
- **应用**
  - ✓ 加密
    - MD5，把一些不同长度的信息转化成杂乱的128/256bits的编码
  - ✓ 信息摘要/查询




```
MD5("version1.2") = "d9671a07d41dc45815aa9e8fbf88148f"

MD5("version2.1") = "51535660790f4243b9cb5dba1a0bc207"
```

词的Signatures

- 一个单词的“签名”是一个**位向量**，由F位组成，其中有m位置1;
- 如下图给出了一段文本，以及文本中部分单词的“签名”示例，其中F=12，m=4;

这 是 一本 关于 信息 检索 的 教材 。 介绍 了 检索 的 基本 技术 。…

文本

单词	词“签名”
技术	001 000 110 010
教材	000 010 101 001
检索	000 000 011 110
信息	101 000 100 001
…	…
…	

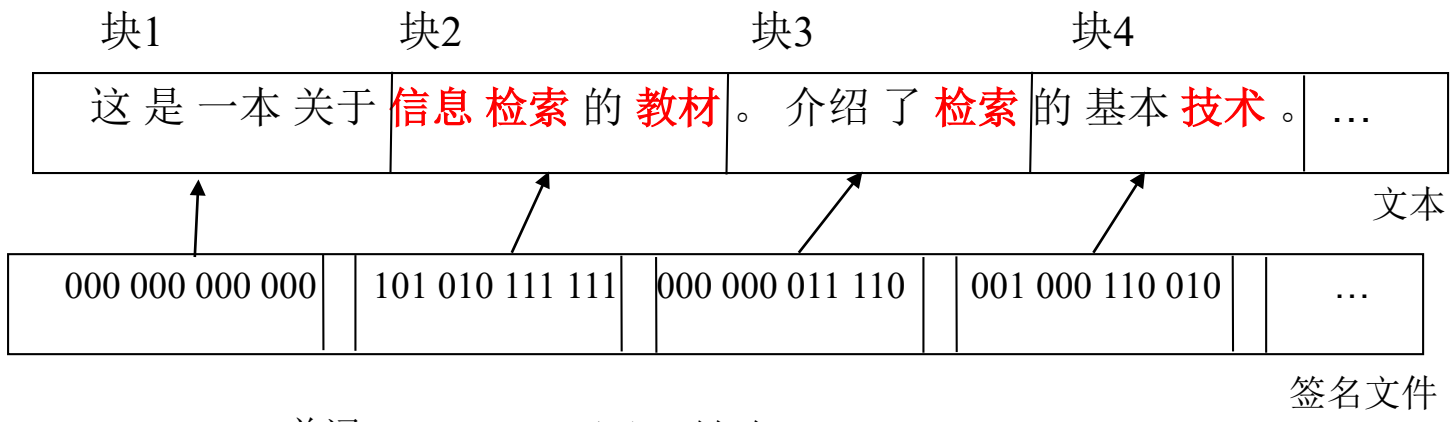
词Signature的生成算法

```
输入：词W，参数F和m
输出：词W的F位“签名”S
算法：
(1) 将W转换为ASCII值，然后转换为32位整数i
      例如：free = 66726565 (hex)
(2) 初始化：
      a) S的F位全置0;
      b) srandom(i);                          //初始化随机种子
      c) j = 0;
(3) while (j < m){
      p = random();                           //生成32位随机数
            p = p mod F;                      //映射到0和F-1之间
      if (S[p] == 0) {                        //确保m位置1
            S[p] = 1;
              j++;
            }
      }
(4) 结束，返回S
```

签名文件

- 如果仅对每一个单词生成相应的“签名”，则在查询的时候，需要逐个比较，非常浪费时间。
- 因此，通常把文本分成若干块，每个块有多个单词，块的“签名”是通过块中单词的“签名”的按位“或”操作得到的
- 因此，所谓的签名文件，就是文本中块“签名”的序列；
- 下图给出了一段文本以及其对应的签名文件

一段文本对应的块签名



单词	词“签名”
技术	001 000 110 010
教材	000 010 101 001
检索	000 000 011 110
信息	101 000 100 001
…	…
…	

000 010 101 001
000 000 011 110
101 000 100 001

块2的签名 101 010 111 111

签名文件的使用和维护

- 使用签名文件检索单个单词的过程是，首先对这个单词使用相同的算法生成其“签名”S，并与所有文本块的签名S_i依次比较, 即计算S & S_i是否与S相等；
- 如果相等，说明S中含有的位， S_i中也有，即S_i文本块可能含有要查找的单词→形成候选文本块
- 对所有候选文本块执行字符串匹配，确定是否真正含有要查找的单词；

块2的签名	101 010 111 111
信息的签名	101 000 100 001
And运算	101 000 100 001

签名文件的使用和维护

- 使用签名文档时，可能会出现误检的情况，即虽然某个单词不存在于文本块中，但是其“签名”仍然能够与该块的“签名”相应位匹配。
- 造成误检的原因有两方面：
  - 主要原因：相匹配的位来自块中不同单词的“签名”。

    例如，假设查询单词“计算机”的“签名”是101 010 010 000，其与块2的“签名”相匹配，但是显然块2中，不含有“计算机”，这时就造成误检，原因就是“计算机”的“签名”101 000 011 000中，1和3位与“信息”的“签名”相匹配，8和9位与“检索”的签名相匹配；
  - 次要原因：**散列冲突**，即两个不同的单词具有相同的“签名”，然而如果F足够大，这种冲突的可能性很低。
- 与倒排文件的维护相比，签名文件的维护非常容易
  - 添加文本时，只需要将文本分块，然后生成块的“签名”，追加到签名文件末尾即可；
  - 删除文本时，只需要在签名文件中删除相应文本块的“签名”即可

块2的签名　　101 010 111 111

分析

- 为了加快检索速度，需要减少顺序匹配的次数，需要将块做得足够大，**一般将一个文本看作是一块**；
- 而为了减少误检的发生率，“签名”的位数又要足够多，经验表明，取文本平均长度的30%~40%为宜；
- 签名文件由于组织简单，因此较容易生成，维护费用较小，它是在倒排文件和全文扫描之间做了空间和时间的平衡，适合于**小文本集合和查询频率较低**的系统
- 签名文件的主要缺点：
  - 索引占用的空间和检索的时间复杂性与原始文档集成线性关系。
  - 误检的发生数也基本和文本集合中的文本个数成正比。
  - 即使对于比较短的查询，也需要大量的磁盘访问操作。
  - 签名文件很难对频率和权重信息进行编码，这就很难支持排序操作。
- **签名文件索引技术只适用于小规模的文本集合。**

文本搜索技术

- 前面的文本检索技术需要事先建立索引，然后才能快速查找。
- 在某些应用中,这种建立索引的方法并不适用
  - 情况1:在签名文件的候选块确认过程中，就需要在块中查找某一查询是否真正存在；
  - 情况2:在文本过滤技术中，一般文本仅需查询一次，这就没必要建立索引；
  - 情况3:在搜索引擎结果后处理中，需要对搜索结果中包含的查询关键词进行加亮显示，也需要用到文本搜索技术;
- 快速的文本搜索非常必要!

文本搜索技术

- 精确的字符串匹配问题可以如下描述:
  - 结定一个长度为m的模式$P(p_1p_2\cdots p_m)$，以及一个长度为n的长文本$T(t_1t_2\cdots t_n)$，其中$n>>m$。在文本T中找出模式P出现的位置。
  - 如：ASDFASPGJAPOTUJGPAFG（T）中PGJA（P）出现的位置。
- 三种常用的精确匹配算法
  - *BF*算法
  - *KMP*算法
  - *BM*算法

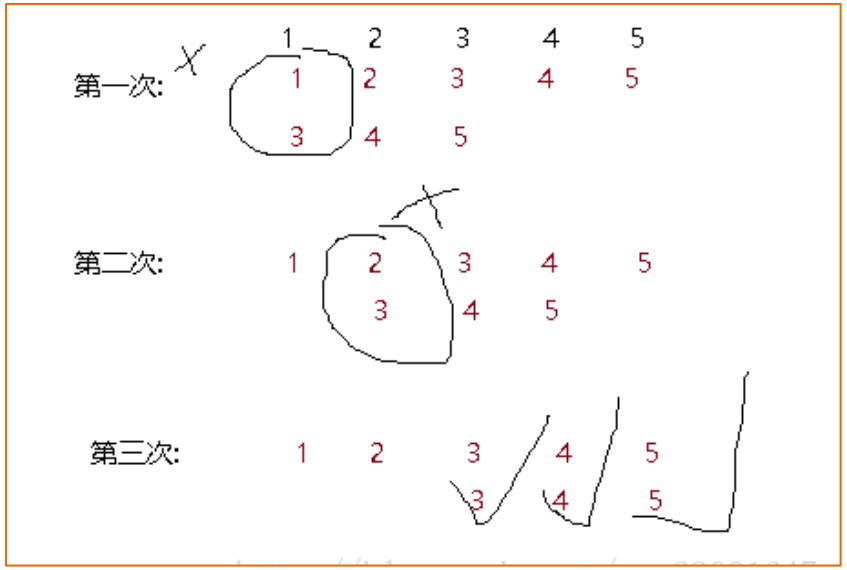
BF算法

- BF算法是**B**rute-**F**orce（蛮力）算法的简称
- 是一种简单、直接、容易实现的字符串匹配算法
- 基本思想：
  - 将模式*P*和文本*T*中的m个字符的子串$t_k t_{k+1} \ldots\ldots t_{k+m-1}$进行匹配，*1<k<n*。
  - 若模式和子串匹配，则返回匹配的位置，
  - 若模式和子串不匹配，则从t_{k+1}位置开始新的考察

BF算法

输入： *文本T和模式P*
输出： *文本T中如果包含模式P返回匹配位置，否则返回匹配不成功*
算法：

```
i = 1; j = 1;
while (i ≤ m and j ≤ n) {
    if (pi == tj) {
        i = i + 1; j = j + 1;
    } else {
        j = j - i + 2; i = 1;
    }
}
if (i > m)
    return 匹配位置j;
else
    return “匹配不成功”;
```



BF算法分析

- 在BF算法中，可以更改循环条件为$j \le n-m+1$，使得循环可以更早地终止（只返回首次位置）；
- 因为存在O(n)个文本位置，在最坏的情况下，验证每个位置需要花费的时间为O(m)，所以BF算法的最长时间为O(mn)
- BF算法的平均时间为O(n)，因为对于一个随机文本，一般进行O(l)次比较之后，就可以发现错误的匹配。
- BF算法无需进行任何形式的预处理，**实现简单，**被大多数程序设计语言所采用
  - 如C语言中的字符串查找函数strstr()使用的就是BF算法
- BF算法的**时间复杂性**与其他算法比较还是比较高的，在对效率要求较高的系统中，应该避免大量使用

KMP算法

- D.E.Knuth、J.H.Morris和V.R.Pratt同时发现的改进的模式匹配算法/克努特—莫里斯—普拉特操作
- 该算法是第一个可以在O(n+m)的时间内完成串模式匹配的算法，**但平均来说**，它的效率并不比BF算法快很多。
- 基本思想
  - 每当匹配过程中出现字符串比较不等时，不像BF算法那样仅将模式向右“滑动”一个位置，而是利用已经得到的“部分匹配”结果将模式向右“滑动”尽可能远的一段距离后，继续进行比较。
  - 可以避免对那些能够推断出不匹配的位置进行徒劳的操作

位置：	1	2	3	4	5	6	7	8	
文本：	a	b	d	a	d	e	f	g	
模式：	a	b	d	f					
		a	b	d	f				← BF 算法仅移动一个字符
				a	b	d	f		← KMP 算法移动三个字符

Donald Ervin Knuth（1938）

KMP算法

- KMP算法原则
  - 从匹配成功的子模式中找出“能够相互匹配的最长的前缀和后缀”
  - 在使用KMP算法进行模式匹配时，需要根据模式事先构造一个shift跳转表，用来决定在某个位置匹配失败时应该移动多少个字符
- Shift表的构造方法
  - 如果当前不匹配的位置为*j*，重复字串的长度为*k*，则跳过的字符个数为*j-k-1*

KMP算法-示例(BF)

```
BBC ABCDAB ABCDABCDABDE
ABCDABD
```

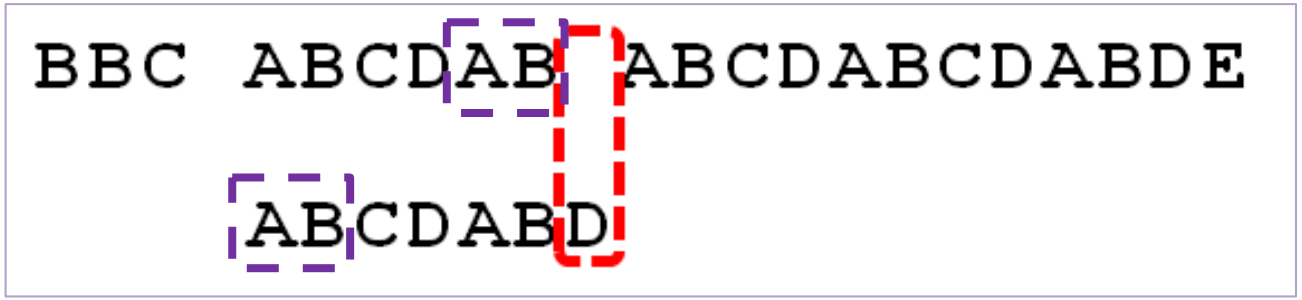
```
BBC ABCDAB ABCDABCDABDE
    ABCDABD
```

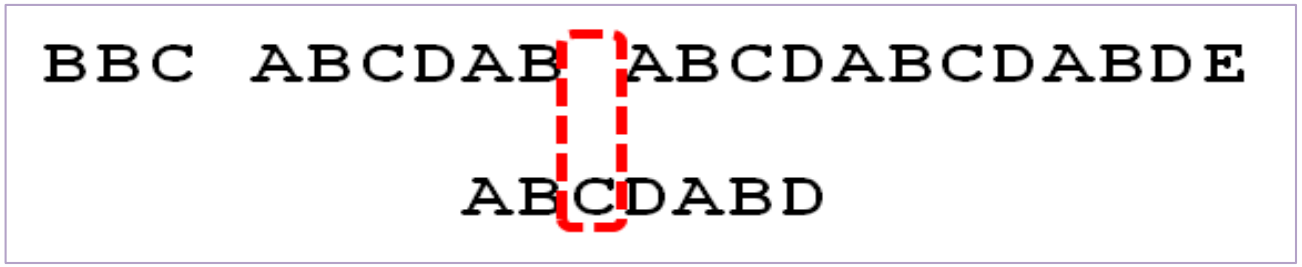
```
BBC ABCDAB ABCDABCDABDE
 ABCDABD
```

```
BBC ABCDAB ABCDABCDABDE
   ABCDABD
```

```
BBC ABCDAB ABCDABCDABDE
    ABCDABD
```

KMP算法-示例(优化)





如果当前不匹配的位置为*j*，**重复字串**的长度为*k*，则跳过的字符个数为*j-k-1*

7-2-1=4

KMP算法的shift表

- 模式P=“abcabcacab”的shift表

匹配失败位置	1	2	3	4	5	6	7	8	9	10
模式字符	a	b	c	a	b	c	a	c	a	b
重复子串长度	0	0	0	0	1	2	3	4	0	1
跳过字符数	1	1	2	3	3	3	3	3	8	8

如果当前不匹配的位置为j,重复字串的长度为k,则跳过的字符个数为j-k-1

- KMP算法的shift表可以在O(m)的时间复杂度下，通过对关键词的分析而获得,m是模式的长度
- KMP算法花费O(m+n)的时间来解决模式匹配问题

BM算法

- **B**oyer和**M**oore提出
- 是另一个与KMP算法截然不同的却同样拥有线性时间复杂度的算法
- BM算法在实际的模式匹配中跳过了很多无用的字符，不必对无用的字符进行匹配（而在KMP算法中，文本串中的每个字符都是需要进行比较操作的）
- 这种跳跃式的比较方式，使BM算法的效率极高，特别是在大字符集上进行字符串的模式匹配时优势更加明显。
- 在理论和实践上，**BM算法都比KMP算法更有效**
- 基本思想
  - 假设模式的长度为m。先令模式和文本左对齐，然后对模式中最右一个字符p_m与其在文本中相对应的字符t_m进行比较。如果比较的结果是不匹配，那么我们接着考察t_m在模式中出现的最右位置，然后根据这个位置移动模式，使其和t_m对齐

BM算法

- 将模式和文本左对齐后，模式中**最后一个字符**与其在文本中对应字符比较的结果有下面两种可能
  - 第一种情况:t_m根本没有在模式中的任何一个位置出现,那么就可以放心大胆地将模式向后移动m个字符,然后将模式中最后一个字符与它现在所对应的文本中的字符(即t_{2m})进行比较.
  - 第二种情况:如果t_m是模式中的第n个字符,那么就可以将模式向后移动m-n个字符.

文本:	A	B	X	**G**	E	F	H	J	…
模式:	B	C	D	**E**					
移动后:					B	C	D	E	

文本:	A	B	X	**C**	E	F	H	J	…
模式:	B	**C**	D	E					
移动后:			B	**C**	D	E			

- *文本字符G*没有在模式中,可以将模式向后移动*m*个字符
- 文本字符C现现在模式中的第2个字符，可以将模式向后移动4-2个字符

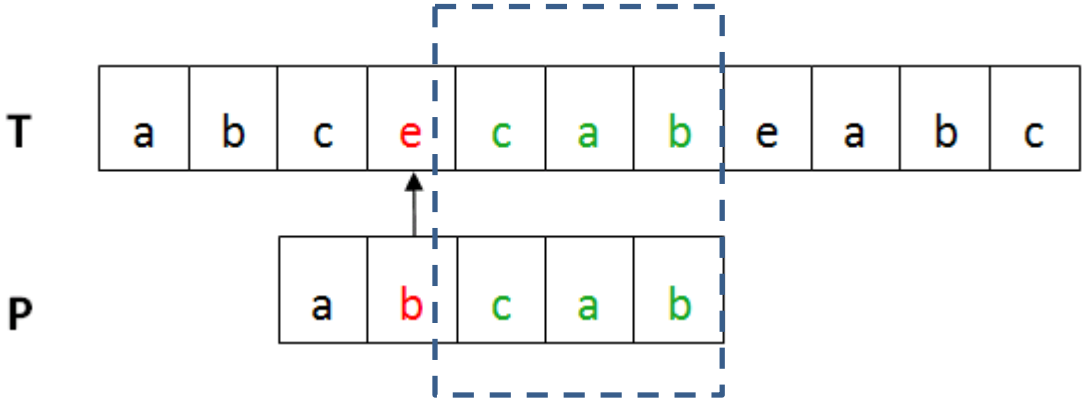
$$skip(x) = \begin{cases} m;\ x \neq P[j](1 \leq j \leq m)\ ，即\ x\ 在\ P\ 中未出现 \\ m - max(x);\ \{k|P[k] = x, 1 \leq k < m\};\ \ x\ 在\ P\ 中出现 \end{cases}$$

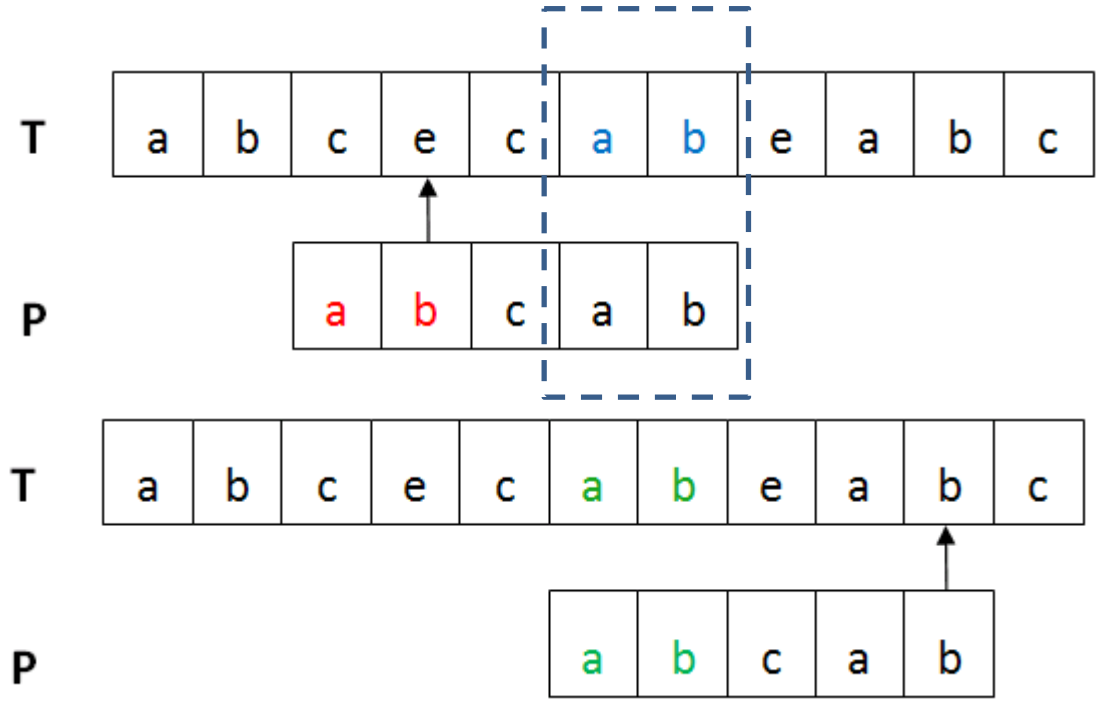
BM算法-好后缀规则

若发现某个字符不匹配的同时，已有部分字符匹配（好后缀）成功，则按如下两种情况讨论：

第一种情况，如果在P中位置t处已匹配部分P'在P中的某位置t'也出现，且位置t'的前一个字符与位置t的前一个字符不相同，则将P右移使t'对应t方才的所在的位置。

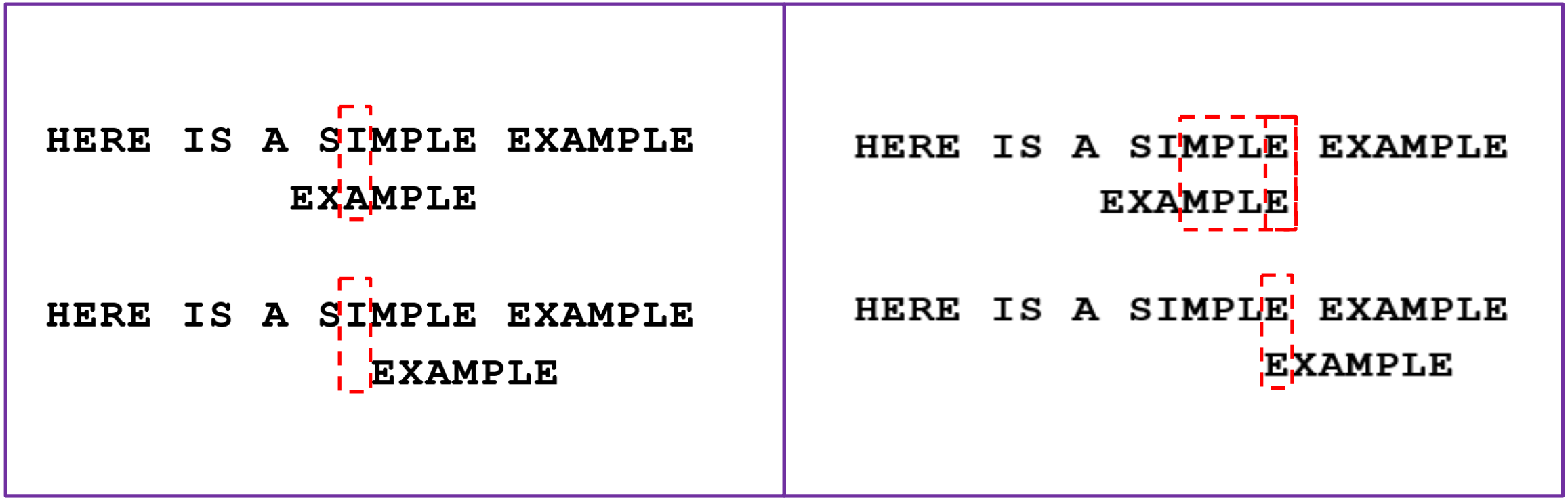
第二种情况，如果在P中任何位置已匹配部分P'都没有再出现，则找到与P'的后缀P''相同的P的最长前缀x，向右移动P，使x对应方才P''后缀所在的位置。





BM算法-选择

好后缀规则与坏字符规则中移动步数较大者作为最后移动步数



模式匹配算法的选择

- 如果模式的长度很小（1到3个字符），可以使用BF算法，因为其实现简单，而且不需要额外构造跳转表
- 如果字母表很大，KMP算法是一个选择，因为模式中含有重复的情况较少
- 除此以外，特别是对于长文本来说，BM算法是最佳选择。

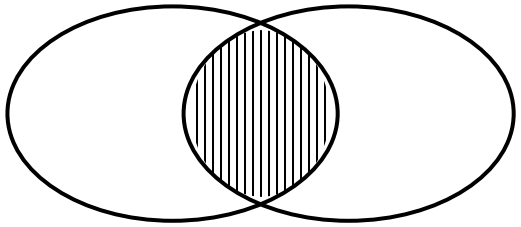
布尔检索

- 布尔逻辑算符及其应用
  - 常用的布尔逻辑算符有三种，分别是逻辑与AND、逻辑或OR、逻辑非NOT，用以表达两个检索词之间的逻辑关系。下面分别简释他们各自的含义与用法。
- （1）逻辑与——“AND”或 *
  - 检索词A与检索词B若用“AND”组配，则提问式可写为“A AND B”或者“A * B”。检索时，数据库中同时含有检索词A和检索词B的文献，才算是命中文献。

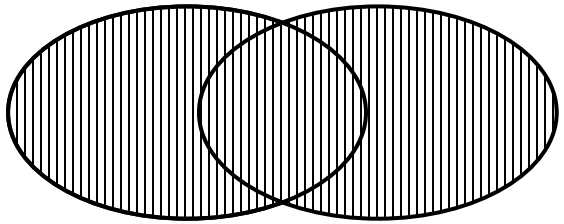
布尔检索

- （2）逻辑或——“OR”或 +
  - 检索词A和检索词B若用“OR”组配，则提问式可写为“A OR B”或者“A + B”。检索时，数据库中的文献凡含有检索词A或者检索词B或者同时含有检索词A和B的，均为命中文献。
- （3）逻辑非——“NOT”或 —
  - 检索词A和检索词B若用“NOT”进行逻辑组配，则可写为“A NOT B”或者“A — B”。对于这个提问式，数据库中凡含有检索词A而不含检索词B的文献，为命中文献。
- （4）逻辑异或 ——”XOR”或⊕（少）
  - 检索词A和检索词B若用异或XOR组配，可写为“A XOR B”或者“A⊕B”。该检索式的检索结果为：含有检索词A的文献命中，含有检索词B的文献命中，但同时含有A和B的文献不命中。

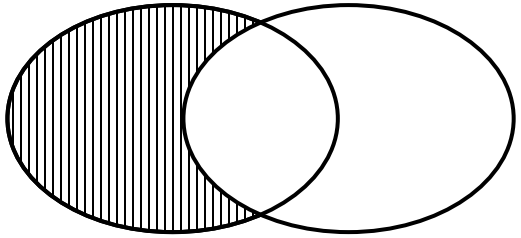
布尔运算示意图



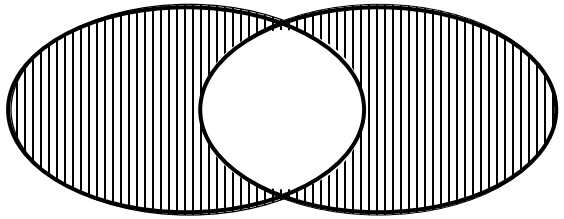
Ⅰ “AND”运算



Ⅱ “OR”运算



Ⅲ “NOT”运算



Ⅳ “XOR”运算

注意事项

- （1）布尔检索执行顺序。三种布尔检索运算符之间的优先顺序为**NOT、AND、OR**。有括号时，先执行括号内的逻辑运算。有多层括号时，先执行最内层括号中的运算。
- （2）不同检索工具的布尔逻辑检索有不同的表现形式，在使用的时候，要注意先了解相关的使用规则。

CNKI 例1

要求检索钱伟长在清华大学或上海大学时发表的文章。

检索式：AU =钱伟长 and (AF =清华大学 or AF =上海大学)

CNKI 例2

要求检索钱伟长在清华大学期间发表的题名或摘要中包含“物理”的文章。

检索式：AU =钱伟长 and AF=清华大学 and (TI =物理 or AB = 物理)

截词检索

- 所谓截词（truncation），是指检索者将检索词在自己认为合适的地方截断；
- 截词检索，则是用截断的检索词的一个局部去数据库中进行检索，凡是能与这个词局部中的所有字符（串）相匹配的文献，即为命中文献。
- 截词符号在不同的信息检索系统中表示不同，但功能是相同的。
- 通常情况下用“*”表示无限截断，用“？”表示有限截断。

后截词检索

后截断是最常用的截词检索技术。将截词符号置放在一个字符串右方，以表示其右的有限或无限个字符不影响该字符串的检索。从检索性质上讲，后截断是**前方一致**检索。

- *【例】coagula**
- *可检出的词汇有：coagula、coagulable、coagulant、coagulase、coagulate、coagulation、coagulative、coagulator等*
- *【例】mold??*
- *可检出的词汇有：mold、molded、molder等，但却不能检出下述词汇：moldery、molding、moldman、moldwash。*

前截词检索

与后截断相对，前截断是将截词符号置放在一个字符串左方，以表示其左的有限或无限个字符不影响该字符串的检索。从检索性质上讲，前截断是**后方一致**检索。

- *【例】* meter*
- *可检出的词汇如下：meter、cubic-meter、macro-meter、macrometer、mini-meter、minimeter、square-meter……，但是检不出meterage、metering、meterman等。*

中截词检索

- 中截断又称为“通用字符法”或“内嵌字截断”或“屏蔽”。这种截断是把截断符置于一个检索词的中间，允许检索词的中间有若干形式的变化。一般地，中截断仅允许有限截断。
- 英语中有些单词的拼写方式有英式、美式之分，有些词则有某个元音位置上出现单复数不同，如：organization↔organisation，man↔men，woman↔women等等。文献数据库中与之相似的例子是大量存在的。若希望不漏检，使用这种词进行检索时就要用中截断的处理方法。
  - 比如，上述词汇在用于检索时可写成：organi? ation，m? n，wom? n。

限制检索

- 字段检索是限定检索词在数据库记录中出现的**字段范围**的一种检索方法。
- 在检索系统中，数据库设置、提供的可供检索的字段通常分为主题字段和非主题字段两大类。每个字段都有一个用两个字母表示的字段代码。
- 在CNKI中各字段代码及对应名称说明如下：
  - SU='主题',TI='题名',KY='关键词',AB='摘要',FT='全文',AU='作者',FI='第一责任人',AF='机构',JN='中文刊名'&'英文刊名', RF='引文',YE='年',FU='基金',CLC='中图分类号',SN='ISSN',CN='统一刊号',IB='ISBN' ,CF='被引频次'

加权检索

- 词加权系统（term weighting system）是最常见的加权检索系统。
- 检索者根据对检索需求的理解选定检索词，同时对提问中的每一个检索词（概念）给定一个数值以表示其重要性程度，即权（weight）。在检索中，先查找这些检索词在数据库记录中是否存在，然后计算存在检索词的记录所包含的检索词的权值总和，通过与预先给定的阈值（threshold）进行比较，权值之和达到或超过阈值的记录视为命中记录，命中记录的输出按照权值总和从大到小排列输出。这种用给检索词加权来表达提问要求的方式，称为词加权提问逻辑。

加权检索

【例】以“住房补贴政策”为检索课题，给检索词“住房”、“补贴”和“政策”分别赋予权值4、5、3，阈值T=5。检索时，在关键词文本框中输入“住房/4*补贴/5*政策/3”，单击查询，则依所含关键词的权重检出相应记录，命中文献按权值递减排列如下：

住房，补贴，政策　　权和=12≥5

住房，补贴　　权和=9≥5

补贴，政策　　权和=8≥5

住房，政策　　权和=7≥5

补贴　　权和=5≥5

词频加权检索

词频加权检索是根据检索词在文档记录中出现的频率来决定该词的权值，而不是由检索者来指定检索词的权值。在这一方面，词频加权就消除了人工干预因素。

TI='转基因$2（CNKI）

小结