信息检索

给定用户需求返回满足该需求信息的一门学科。通常涉及信息的获取、存储、组织和访问。是从大规模非结构化的数据（通常是文本）的集合中找出满足用户信息需求的资料的过程

结构化数据是指数据库表中的数据，数据库常常支持范围或者精确匹配查询。

非结构化的数据通常指的是自由文本，允许进行关键词加长操作符号的查询；也允许进行更复杂的概念性查询。

半结构化数据是指没有数据是完全无结构的，比如，网页数据

信息检索中的基本假设

（1）文档集Collection：由固定数目的文档组成

（2）目标：返回与用户需求相关的文档并辅助用户完成某项任务

检索结果的评价

（1）正确率：返回的结果中真正和需求相关的文档所占的百分比

（2）召回率：所有和信息需求真正相关的文档中被检索系统返回的百分比

布尔检索

针对布尔查询的检索，布尔查询是指利用AND，OR或者NOT操作符将词项连接起来的查询。

问题：莎士比亚的哪部剧本包含Brutus及Caesar但是不包含Calpurnia？

（1）暴力方法：从头到尾扫描所有剧本，对每个剧本进行判断

暴力方法的不足包括：

a.查询速度超级慢

b.处理not语句不方便

c.不太容易支持其他操作

d.不支持检索结果的排序

但是暴力方法实现简单，且支持文档的动态变化

（2）词项-文档（term-doc）关联矩阵

若某个剧本中包涵某个单词，则在矩阵的对应位置上置1，否则置0

关联向量（incidence vectors）：

关联矩阵的每一列（对应每一篇文档）都是0/1向量，每个0/1都对应一个词项。

关联举证的每一行（对应一个词项）也是一个0/1向量，每个0/1代表该词项在相应文档中是否出现。

对于上述问题，可以取三个词对应的行向量，进行按位与操作。

词项-文档矩阵的不足：当文本数目较大时，矩阵将非常大。词项文档矩阵是稀疏矩阵，我们可以通过只记录1的位置进行优化。

（3）倒排索引(Inverted index)

对于每个词项t，记录所有包含t的文档列表。

每篇文档用一个唯一的docID来表示，通常是正整数，如1，2,3

倒排索引中的名词解释：

a.词项词典：简称词典，表示每一个需要索引的单词

b.倒排记录表，又称倒排表，每个词项都有一个记录出现该词项的所有文档的列表，该表中的每个元素记录的是词项在某文档中的一次出现信息。这个表中的每个元素通常称为倒排记录。每个词项的倒排记录表一起构成全体倒排记录表。

> 倒排记录表会按照docID进行排序，这是高效进行查询处理的关键。

c.文档频率：出现某词项的文档的数目，就是每个倒排记录的长度。此信息对于倒排索引并不是必须的，但是可以在处理查询时提高搜索的效率。

> 排序检索模型中会频繁使用文档频率。

建立倒排索引的最核心的步骤就是将这个列表按照词项的字母顺序进行排序，其中一个词项在同一文档中的多次出现会合并在一起。最后整个结果分成词典和倒排记录表两部分。

在最终得到的倒排索引中，词典和倒排记录都有存储开销，词典放在内存中，倒排记录表由于存储开销巨大， 一般放在磁盘上。

对于内存中的一个倒排记录表，通常有两种比较好的存储方式，一是单链表，一是变长数组。

布尔查询的处理：

对于上述问题，我们需要在词典中分别定位Brutus及Caesar，并返回他们的倒排记录表，然后对倒排记录表求交集。

交集操作：又称合并操作，需要对每个有序列表都维护一个位置指针，并让两个指针同时在两个列表中后移。每一步比较两个位置指针所指向的文档ID，如果两者一样，则直接输出该ID到结果表中，然后同时将两个指针后移一位；如果两个文档ID不同，则将较小的ID所对应的指针后移。

假设两个倒排记录表的大小分别是x和y，则上述求交集的过程需要O(x+y)次操作。

> 这个算法对于每个倒排记录表来说都是线性的。

查询优化：

如何通过组织查询的处理过程来使处理过程中工作量最小。

对于布尔查询进行优化需要考虑的一个主要因素是倒排记录表的访问顺序=>需要考虑文档频率

按照词项的文档频率从小到大依次进行处理，如果先对两个最短的倒排记录表进行合并，那么所有中间结果的大小都不会超过最短的倒排记录表。

此外，布尔查询并不是单纯的输入不同长度的倒排记录表输出最后合并结果的过程，对于布尔查询而言，其是将每个返回的倒排记录表和当前内存中的中间结果进行合并。