搜 索 技 术 研 讨

目录/CONTENTS

01

引擎分类与概述

引 擎 分 类 与 概 述

搜索引擎按其工作方式主要可分为四种，分别是全文搜索引擎（Full Text Search Engine）、垂直搜索引擎（Vertical Search Engine）、元搜索引擎（Meta Search Engine）和目录搜索引擎（Catalogue Search Engine）。

全文搜索引擎是名副其实的搜索引擎，欧美具代表性的有Google、Fast/AllTheWeb、 AltaVista、Inktomi、Teoma、WiseNut等，中国著名的有百度（Baidu）。它们都是通过从互联网上提取各个网站的信息（以网页文字为主）而创建的数据库。检索与用户查询条件匹配的相关记录，然后按一定的排列顺序将结果返回给用户，因此他们是真正的搜索引擎。

垂直搜索引擎是针对某一个行业的专业搜索引擎，是搜索引擎的细分和延伸，是对网页库中的某类专门的信息进行一次集成，定向分字段抽取出需要的数据进行处理后再以某种形式返回给用户。垂直搜索是相对通用搜索引擎的信息量大、查询不准确、深度不够等提出来的新的搜索引擎服务模式，通过针对某一特定领域、某一特定人群或某一特定需求提供的有一定价值的信息和相关服务。例如，著名的百度图片搜索，互联统计网等都是针对某一领域而采用的垂直搜索引擎。

元搜索引擎在接受用户查询请求时，同时在其他多个引擎上进行搜索，并将结果返回给用户。著名的元搜索引擎有InfoSpace、Dogpile、Vivisimo等（元搜索引擎列表），中文元搜索引擎中具代表性的有搜星搜索引擎。在搜索结果排列方面，有的直接按来源引擎排列搜索结果，如Dogpile，有的则按自定的规则将结果重新排列组合，如Vivisimo。

目录搜索引擎是以人工方式或半自动方式搜集信息，由编辑员查看信息之后，人工形成信息摘要，并将信息置于事先确定的分类框架中。信息大多面向网站，提供目录浏览服务和直接检索服务。

————引用自《维基百科》

02

基础检索算法

基 础 检 索 算 法

在了解搜索引擎的核心技术前，我们需要对我们现阶段能掌握的基础检索算法进行了解。
首先我们可以把检索抽象地看成这样一个过程，该过程确定一个具有某特定值的元素是否是某集合的成员。而精确匹配查询是指检索关键码值与某个特定值匹配的记录。范围查询则是指检索关键码在某个特定值范围内的所有记录。
由此，我们把检索算法可以大致分为三类：
1.顺序表和线性表方法。
2.根据关键码值直接访问方法（散列法）。
3.树索引方法。
由于树索引方法可以参照BST进行理解，又或者参照B+树进行理解，在此不展开讨论。

基 础 检 索 算 法

1.顺序表和线性表方法

我们对于一般在线性表中查找元素的方法已经比较熟悉，可以用最基础的顺序查找法，也可以用效率更高的二分检索法。在这些算法的基础上，我们延伸出**自组织线性表**的概念。这种算法好处在于我们可以通过估算的访问频率排列记录，利于快速查找，有点类似于哈夫曼树的思想。

由于访问频率是不断改变的，所以我们需要启发式规则决定如何重新排列线性表。以下是三个传统的启发式规则：

1）计数方法：为每一条记录保存一个访问计数，而且一直按照这个顺序维护记录。

2）移至前端方法：如果找到一条记录就把它放到线性表的最前面，而把所有记录后退一个位置。

3）转置：把找到记录与它在线性表中的前一条记录交换位置。

例如我们访问一组记录：F D F G E G F A D F G E

计数方法组织，经过访问后，线性表结果：F G D E A B C H

而移至前端方法组织：E G F D A B C H

而转置组织：A B F D G E C H

2.根据关键码值直接访问方法（散列法）

我们知道数组的特点是：寻址容易，插入和删除困难。而链表的特点是：寻址困难，插入和删除容易。那么我们能不能综合两者的特性，做出一种寻址容易，插入删除也容易的数据结构？这就是**散列表**，也称为哈希表。

我们通过关键码直接访问表，可以把关键码值映射到表中的位置来访问记录，这个过程我们称之为散列。这个映射函数叫做散列函数，存放记录的数组叫做散列表。散列搜索一般适合于精确匹配查询。

散列表查找关键码值K的记录步骤如下：

1）计算表的位置h（K）

2）从槽h（K）开始，使用冲突解决策略找到包含关键码值K的记录

而我们使用散列表的初衷是优化查找速度，所以为了方便计算，减少堆积，我们设计时需要意识到计算不能太复杂，散列地址均匀分布这两点。由此引申出一下散列策略：

1）直接地址法

2）数字分析法

3）平方折中法

4）折叠法

5）除留余数法

6）随机数法

在此不再展开赘述。

03

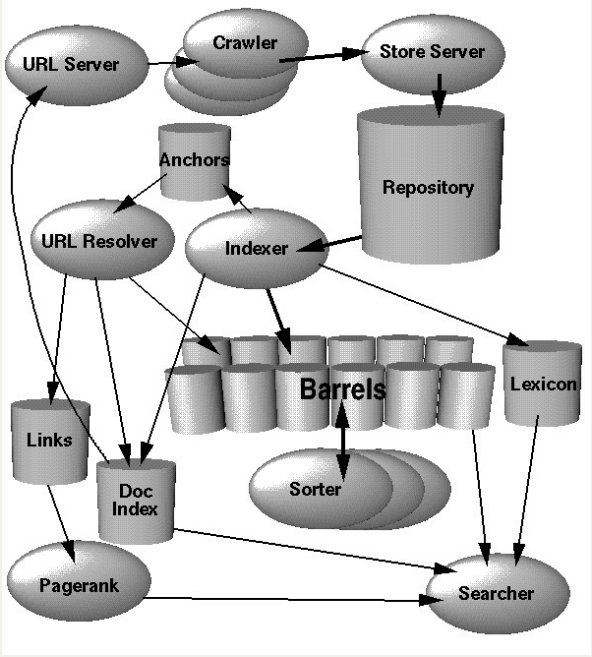
搜索引擎核心技术

搜索引擎核心技术

由于现在计算机应用程序都以大型数据库为中心，在检索技术之上进行整合，诞生了索引技术，索引技术大概分为线性索引，ISAM（线性索引的升级），树型索引。但这些都是广为人知的索引技术，而我们以谷歌为例子，探讨其真正的核心技术。

假设我们自己进行简单的搜索，我们能通过python写爬虫，爬数据，然后用开源的全文搜索引擎进行架设，我们自己的搜索引擎就算完成了。看上去技术含量不大，那么谷歌是如何做到在搜索领域独步全球？

我们先来看谷歌搜索引擎的整体架构：



搜索引擎核心技术

在Google中，网络爬行（网页的下载）是由很多分布的爬行器(crawlers)来做的。由一个URL服务器(URLserver)来向爬行器发送URL列表。被抓取下来的网页就被送到存储服务器(store server)上。存储服务器再把网页压缩并存储在仓库(repository)里。每一篇网页都有唯一一个与之相关联的ID号，称作docID，每当有一个新的URL被分析出来的时候都会被赋予一个docID。索引函数由索引器(indexer)和整理器(sorter)在执行。索引器会执行若干个函数。它会读取仓库、解压文档并分析它们。每一篇文档都被转化为词汇出现情况的集合，被称为hits。hits记录了词汇、在文档中的位置、对字号的估计和大小写。索引器将这些hits分发到一些“桶”(barrels)的集合中，创建部分整理好的前向索引。索引器还要执行另一个重要的函数。它要分析出每一篇网页中的链接然后将关于它们的重要信息存储在一个链接(anchors)文本文件中。这个文件包含了足够的信息，可以判断每个链接分别是从哪里到哪里，还有链接上的文本。

URL分解器(URLresolver)读取链接文件(anchors)，并将其中相对URL转化为绝对URL，然后转化成docID。它将链接文本放入前向索引中，将链接所指向的docID与之关联起来。它还要生成一个docID键值对的链接数据库(links)。链接数据库是用来给所有文档计算PageRank的。

整理器(sorter)获得以docID整理过的桶(barrels)，再根据docID进行整理，生成倒排索引。为了使这个操作几乎不需要临时空间，这一步要在一个特定的地方执行。整理器在倒排索引中产生wordID和偏移量的列表。一个叫DumpingLexicon的程序将这个列表和由索引器生成的词典(lexicon)比较构造出搜索器所用的新词典。搜索器由网络服务器运行，利用由DumpingLexicon构造的词典和倒排索引还有PageRank来回应查询。

搜 索 引 擎 核 心 技 术

或许你觉得大段文字流程看下来过于痛苦，我们可以拆分成以下几个部分来分析。

1.爬行网络。为了覆盖上亿篇网页，Google有一个快速的分布式爬虫系统。一台URL服务器给数台（我们一般运行三台）爬虫提供URL。URL服务器和爬虫都是用 Python实现的。每台爬虫一般同时维护300左右个的连接。只有这样才能使网页抓取速度有足够快的速度。在峰值速度下，四个爬虫的系统每秒钟能爬行上百篇网页。这就是说数据率在每秒600K左右。DNS解析是一个主要的性能瓶颈。每个爬虫都自己维护一个DNS缓存，这样就不必在爬取每篇文档前都去解析 DNS。上百个连接中的每一个都可能处于几种不同的状态中：DNS解析、连接主机、发送请求和接受回应。这些因素使得爬虫程序成为系统中的一个复杂的组成部分。它使用异步IO来处理事件，还要有许多队列来转换页面抓取的状态。**（概括，能快速获取数据）**

2.索引网络。

分析——任何被设计为运行在整个网络上的分析器都必须能够处理大量可能出现的错误。

将文档索引到桶中——在分析完每一篇文档后，它都被编码放进一些桶中。每个词都被一个内存中的散列表（词典转化成wordID）。有新词加入词典散列表时，会记录到日志文件中。一旦词汇都被转化为wordID后，它们在当前文档中的出现情况就会被转换成hit列表并被写入到正排桶中。

整理——为了产生倒排索引，整理器将每个正排桶按照wordID排序并分别为标题、链接、和全问生成倒排桶。一次只有一个桶被处理，所以它需要的临时存储空间很少。另外，我们将排序步骤并行化，我们只要简单的运行多个整理器就能尽可能地用上所有的机器，这样就能同时处理多个桶。

（概括，能快速分析分类整理数据）

3.排序。Google比一般的搜索引擎对网络文档维护了更多的信息。每一个hit列表包含了位置、字体和大小写信息。另外，我们还考虑了链接文字里的hit和文档的PageRank的影响。在排序中综合考虑这些信息是很困难的。我们对排序函数设计是没有那个因素的影响过大。首先考虑最简单的情况——单一词汇的查询。为了给单一词汇查询中的一篇文档评级，Google先在这篇文档的hit列表找到中那个词。Google假设每一个hit都属于某种类型（标题、链接、URL、大字号普通文本、小字号普通文本……），每种类型都有自己的权重。这些权重构成了一个由类型索引的向量。Google将hit列表中每种类型的hit计数。然后将计数值转换为计数 - 权重。计数 - 权重先随着计数值线性增长但是很快就停止增长了，计数超过某个值时就与计数没有关系了。我们将计数 - 权重与类型 - 权重向量的内积作为该文档的IR得分。最后，由结合IR和分和PageRank给出该文档最后的评级。

对于多词查询，情况就复杂多了。多个hit列表必须同时扫描，才能使同一文档中相邻的hit的权值比离得远的hit的权值高。分布在不同hit列表中的 hit分开匹配，以便相邻的hit一起匹配。对于每个匹配hit集，都要计算一个邻近度。这个邻近度基于hit在文档（或者链接）中的距离计算，但是被分为十个等级，范围从短语匹配到“毫不相干”。我们不仅对每种类型的hit计数，而且还对每种类型和邻近度计数。计数被转换成计数 - 权重，然后我们取计数 - 权重和类型—邻近度—权重的内积计算IR得分。所有这些数和矩阵都可以在一种特殊的调试模式下被显示出来。这些显示输出对排序系统的开发工作大有帮助。

（概括，能将数据权重化进行排序）

搜索引擎核心技术

此处我们额外介绍一下Google为了产生高精度结果，利用网络的链接结构来计算网页的排名值，称为**PageRank**的算法。PageRank算法根据加权系数，推断该其他链接到网页的价值来处理。PageRank如此获取由人所创建的链接，与及与人关系的重要性。先前的排名搜索方法，采用了许多搜索器，以搜索的关键词和何时搜索来排名页面，或有多相关地关系该搜索。另外，Google亦采用其他秘密准则，决定排名网页的结果。

PageRank并非对指向网页的链接平等的计数，而是将链接数标准化。我们假如有网页 T1...Tn指向网页A，指数d是0到1之间的抑制因子(damping factor)，通常设为0.85。同时C(A)定义为从网页A指出的链接数。网页A的PageRank值如下：

PR(A) = (1-d) + d(PR(T1) / C(T1) + ... + PR(Tn) / C(Tn))

注意，所有的PageRank再各网页中形成概率分布，所以所有网页的PageRank的和会是1。

1. 分析查询词。
2. 将词汇转换为 wordID。
3. 在短桶中对每条词汇查找到 doc 列表的开头。
4. 扫描整个 doc 列表，直到有一篇文档匹配所有搜索项。
5. 为查询计算该文档的排序值。
6. 如果我们在短桶中，并且在任何 doc 列表尾部，就在全文桶中对每一条词汇查找到 doc 列表起始处，然后跳到第 4 步。
7. 如果我们不在任何 doc 列表的尾部，跳到第 4 步。
8. 根据排序值将匹配文档排序，然后返回前 k 个结果。

【表 4：Google 查询评价】

文　献　参　考

想要了解更多，可以查看谷歌给出的原理文章
http://infolab.stanford.edu/~backrub/google.html
或者参考国内粗译过的文章的部分
https://blog.csdn.net/fover717/article/details/8081131
前文涉及的相关知识摘自
维基百科
The Anatomy of a Large-Scale Hypertextual Web Search Engine
————Sergey Brin and Lawrence Page
《数据结构与算法分析》————Clifford A. Shaffer
《大话数据结构》————程杰

THANKS FOR WATCHING