搜索引擎核心技术与算法 —— 倒排索引初体验

原创 QvQ 夕小瑶的卖萌屋 2019-12-26

今天开启一个新篇章——智能搜索与NLP。本篇章将由羸弱菜鸡小Q和大家一同学习与智能搜索相关的知识和技术，希望能和大家一同学习与进步，冲鸭！！

这里首先区分两个概念：**搜索和检索**

检索：数据库时代的概念，及将数据存入数据库，有需要的时候进行查取。对结果的要求绝对精确；比如我要在图书馆里找到所有出现“白马”字样的图书，这里用到的就是检索。

搜索：互联网时代的概念，人们将信息资源放在网上，第三方将互联网的信息搜罗起来，建立索引，所以搜索更多是指基于问题相关性的信息收集方式。当我想知道“如何骑白马最帅？”的时候，这里就不能只罗列出现过“白马”字样的图书了，毕竟骑白马的不一定是王子，还有可能是唐僧 (ノ -_-)ノ ~┻┻。所以这时候就需要真正的搜索了。

闻道有先后，术业有专攻。那我们就先来讲一讲关于检索的那些事儿。

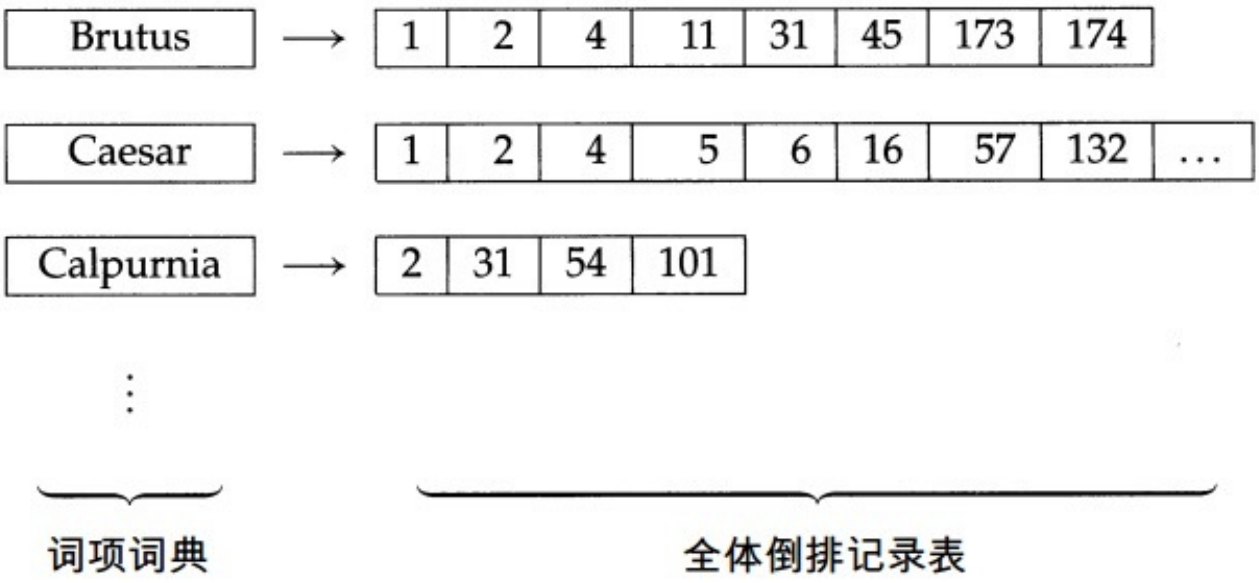
一、倒排索引の初体验

让我来先进入一个例子：我想要在《莎士比亚全集》中找到所有出现过“Brutus”和“Caesar”的章节，一个办法就是从头到尾阅读这本书，留意每一个出现“Brutus”和“Caesar”的地方。这种线性扫描就是一种最简单的计算机文档检索方式。那我现在想找出所有出现过“Calpurnia”章节，难道还是需要对整本书重新阅读一遍嘛？这样时间开销也太大了吧！！！为了解决这个问题，我们引入第一个核心概念——倒排索引。

一个典型的倒排索引分为两部分：**词项词典和倒排记录表**（如下图所示）

每一个词项对应一个倒排记录，一条倒排记录表示该词项所出现过的所有文档列表（如“Brutus”在第1、2、4、11...篇文档中出现过），文档列表的可以是无序的，但是升序更有利于我们之后的高效检索，下文会提到。

易得，建立一个倒排索引的时间复杂度是O（N），其中N是所有文档中单词的数量。



图一 倒排索引的两个部分

那么有了倒排索引后，我们如何完成上诉检索任务呢?

先以一个简单的查询为例：找到同时出现“Brutus”和“Calpurnia”两个单词的文档。

具体步骤如下：

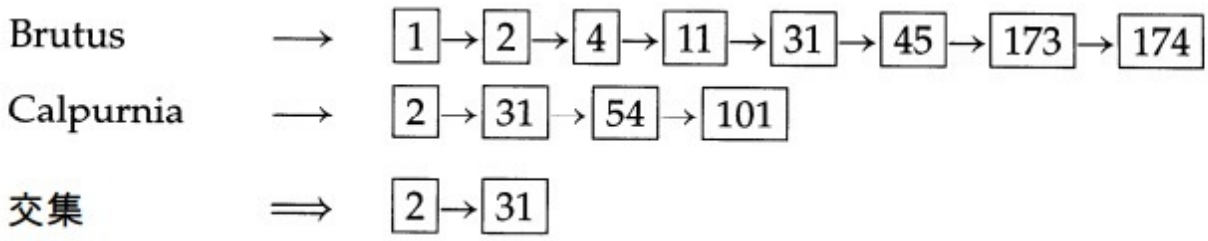
(1)在词典中定位Brutus;

(2)返回其倒排记录表;

(3)在词典中定位Calpurnia;

(4)返回其倒排记录表;

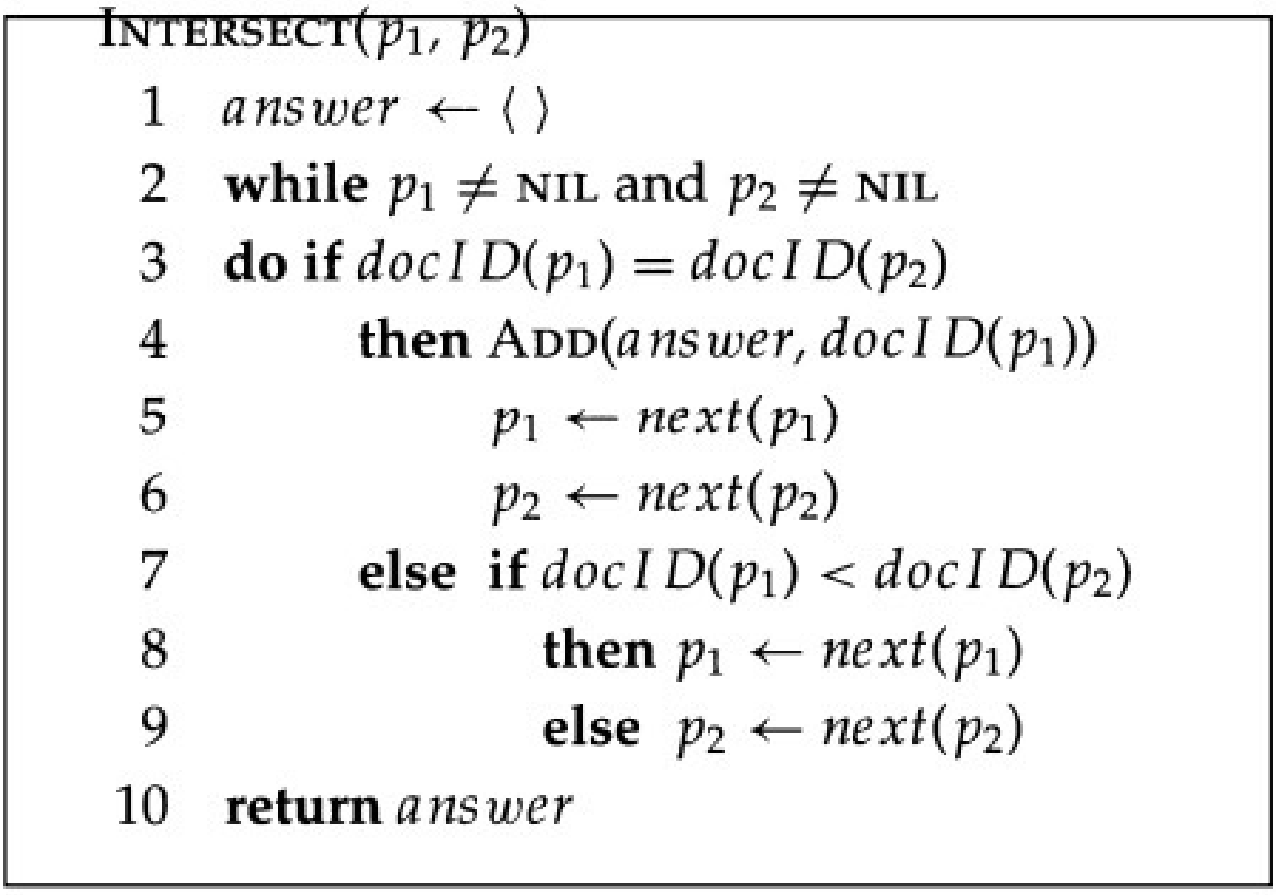
(5)对两个倒排记录表求交集，如下图



图二 求交集示意图

在这里，交集（intersection）操作非常关键，这是因为我们必须快速将倒排记录表求交集以尽快找到哪些文档同时包括两个词项。该操作有时也称为合并（merge）。经常刷题的同学一眼就看出来，这不就是有序数组求交集嘛？leetcode原题实锤了嗷！（leetcode 1213 原题，easy难度）⊙﹏⊙|||

伪代码如下：



```
INTERSECT(p1, p2)
 1  answer ← ⟨ ⟩
 2  while p1 ≠ NIL and p2 ≠ NIL
 3  do if docID(p1) = docID(p2)
 4        then ADD(answer, docID(p1))
 5             p1 ← next(p1)
 6             p2 ← next(p2)
 7        else if docID(p1) < docID(p2)
 8                then p1 ← next(p1)
 9                else p2 ← next(p2)
10  return answer
```

图三 merge伪代码

思路就是用两个指针p1、p2分别指向两个倒排记录（倒排记录是按升序排列的）的头部，如果两个指针指向的文档ID相同，则将该文档放入结果集中，并同时后移两个指针；如果不相同，则将指向文档ID较小的指针向后移动。

```
a = [1, 2, 3, 6, 9, 11, 45, 67]
b = [4, 6, 13, 45, 69, 98]

i = j = 0
result = []
while i < len(a) and j < len(b):
    if a[i] == b[j]:
        result.append(a[i])
        i = i + 1
        j = j + 1
    elif a[i] < b[j]:
        i = i + 1
    else:
        j = j + 1
print(result)  #[6, 45]
```

假设两个倒排记录表的大小分别是x和y，那么上述求交集的过程需要　O(x+y)次操作。 更正式的说法是，查询的时间复杂度为O(N)。和之前“再读一遍书”的扫描方法相比，同样是O(N)的复杂度，但是该方法的中的N是文档数，而前者的N是单词数，假设平均一篇文档有1000个单词，那么倒排索引的效率至少也提高了1000倍。

二、构建倒排索引

为获得检索速度的提升，就必须要事先建立索引。建立索引的主要步骤如下。

(1)**收集需要建立索引的文档，如：**

Doc1: Friends Romans countrymen.

Doc2: So let it be with Caesar …

(2)**将每篇文档转换成一个个词条的列表**，这个过程通常称为词条化（tokenization）或者分词，如：

Friends, Romans, countrymen, So, …

(3) **进行语言学预处理**，产生归一化的词条来作为词项，如：

Friends， roman， countrymen ，So …

(4) 对所有文档按照其中出现的词项来**建立倒排索引**，索引中包括一部词典和一个全体倒排记录表。

> 建立索引最核心的步骤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。词典中同样可以记录一些统计信息，比如出现某词项的文档的数目，即文档频率，这里就是指每个倒排记录表的长度。
> 在最终得到的倒排索引中，词项词典和倒排记录表都有存储开销。前者往往放在内存中，而后者由于规模大得多，通常放在磁盘上。因此，两部分的大小都非常重要。

那么倒排记录表一般利用哪一种数据结构进行存储呢?

由于有些词在很多文档中出现，而另外一些词出现的文档数目却很少，所以，如果采用定长数组的方式将会浪费很多空间。对于内存中的一个倒排记录表，可以采用两种好的存储方法：

（1）**单链表**（singly linked list）便于文档的插入和更新（比如，对更新的网页进行重新采集），因此通过增加指针的方式可以很自然地扩展到更高级的索引策略。

（2）**变长数组**(variable length array)的存储方式一方面可以节省指针消耗的空间，另一方面由于采用连续的内存存储，可以充分利用现代计算机的缓存（cache）技术来提高访问速度。额外的指针在实际中可以编码成偏移地址融入到表中。如果索引更新不是很频繁的话，变长数组的存储方式在空间上更紧凑，遍历也更快。

三、布尔查询及性能优化

在第一章中——找到同时出现“Brutus”和“Calpurnia”两个单词的文档，这个任务其实就是一个布尔查询。

布尔检索模型：接受布尔表达式查询，即通过AND、OR及NOT等逻辑操作符将词项连接起来的查询。在该模型下，每篇文档只被看成是一系列词的集合。例如一个标准的布尔查询可以表示为：

（Brutus OR Calpurnia）AND NOT Caesar

在很多情况下，不论是由于查询语言本身的性质所决定，还是仅仅由于这是用户所提交的最普遍的查询类型，**查询往往是由纯“与”操作构成的**。正是因为这种特性的存在，我们可以进一步优化查询速度。

查询优化指的是如何通过组织查询的处理过程来使处理工作量最小。对布尔查询进行优化要考虑的一个主要因素是倒排记录表的访问顺序。那么，哪种访问顺序具有最优性呢?

一些工程优化策略：

（1）**对每个词项，我们必须取出其对应的倒排记录表，然后将它们合并。**一个启发式的想法是，按照词项的文档频率（也就是倒排记录表的长度）从小到大依次进行处理，如果我们先合并两个最短的倒排记录表，那么所有中间结果的大小都不会超过最短的倒排记录表，这样处理所需要的工作量很可能最少。

（2）在多个布尔检索合并中，**不是将倒排记录表合并看成两个输入加一个不同输出的函数**，而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，这样做的效率更高而最初的中间结果中可以调入**最小文档频率的词项（即该词对应的文档数目最小）**所对应的倒排记录表。算法如下图所示：

```
INTERSECT(⟨t_1, ..., t_n⟩)
1  terms ← SORTBYINCREASINGFREQUENCY(⟨t_1, ..., t_n⟩)
2  result ← postings(first(terms))
3  terms ← rest(terms)
4  while terms ≠ NIL and result ≠ NIL
5  do result ← INTERSECT(result, postings(first(terms)))
6     terms ← rest(terms)
7  return result
```

图四 利用中间结果进行合并

（3）在很多情况下，中间结果表可能会短一个甚至多个数量级。这时可以通过在长倒排记录表中对中间结果表中的每个元素进行二分查找也可以实现合并，使得时间复杂度下降为O(alogb)，其中a、b分别是2个倒排记录的长度。

> 这里就有同学要问了，之前的复杂度不是O(a+b)??这难道是下降嘛?—— 因为中间结果表很短，意味着a << b的情况是可能发生的，所以这里的线性复杂度并不一定要优于多项式对数复杂度。

或者将长倒排记录表用哈希方式存储，这样对中间结果表的每个元素，就可以通过常数时间而不是线性或者对数时间来实现查找。

（4）对于中间结果表，合并算法可以就地对失效元素进行破坏性修改或只添加标记。破坏性修改更适合单链表结构的倒排记录表，而添加标记更适合数组形式。

好了，通过这篇文章，小Q和大家一起对倒排索引在检索中是如何使用的有了一个大体的认识。但**搜索引擎之路道阻且长**，下一篇文章我们将会继续学习词项集合的确定和另外高阶版本的倒排记录表。

参考文献:

《信息检索导论 修订版》

倒排索引优化 - 跳表 —— 博客园 海鸟