53讲算法实战（二）：剖析搜索引擎背后的经典数据结构和算法

像百度、Google这样的搜索引擎，在我们平时的工作、生活中，几乎天天都会用到。如果我们把搜索引擎也当作一个互联网产品的话，那它跟社交、电商这些类型的产品相比，有一个非常大的区别，那就是，它是一个技术驱动的产品。所谓技术驱动是指，搜索引擎实现起来，技术难度非常大，技术的好坏直接决定了这个产品的核心竞争力。

在搜索引擎的设计与实现中，会用到大量的算法。有很多针对特定问题的算法，也有很多我们专栏中讲到的基础算法。所以，百度、Google这样的搜索引擎公司，在面试的时候，会格外重视考察候选人的算法能力。

今天我就借助搜索引擎，这样一个非常有技术含量的产品，来给你展示一下，数据结构和算法是如何应用在其中的。

整体系统介绍

像Google这样的大型商用搜索引擎，有成千上万的工程师，十年如一日地对它进行优化改进，所以，它所包含的技术细节非常多。我很难、也没有这个能力，通过一篇文章把所有细节都讲清楚，当然这也不是我们专栏所专注的内容。

所以，接下来的讲解，我主要给你展示，如何在一台机器上（假设这台机器的内存是8GB， 硬盘是100多GB），通过少量的代码，实现一个小型搜索引擎。不过，麻雀虽小，五脏俱全。跟大型搜索引擎相比，实现这样一个小型搜索引擎所用到的理论基础是相通的。

搜索引擎大致可以分为四个部分：**搜集**、**分析**、**索引**、**查询**。其中，搜集，就是我们常说的利用爬虫爬取网页。分析，主要负责网页内容抽取、分词，构建临时索引，计算PageRank值这几部分工作。索引，主要负责通过分析阶段得到的临时索引，构建倒排索引。查询，主要负责响应用户的请求，根据倒排索引获取相关网页，计算网页排名，返回查询结果给用户。

接下来，我就按照网页处理的生命周期，从这四个阶段，依次来给你讲解，一个网页从被爬取到最终展示给用户，这样一个完整的过程。与此同时，我会穿插讲解，这个过程中需要用到哪些数据结构和算法。

搜集

现在，互联网越来越发达，网站越来越多，对应的网页也就越来越多。对于搜索引擎来说，它事先并不知道网页都在哪里。打

个比方来说就是，我们只知道海里面有很多鱼，但却并不知道鱼在哪里。那搜索引擎是如何爬取网页的呢？

搜索引擎把整个互联网看作数据结构中的有向图，把每个页面看作一个顶点。如果某个页面中包含另外一个页面的链接，那我们就在两个顶点之间连一条有向边。我们可以利用图的遍历搜索算法，来遍历整个互联网中的网页。

我们前面介绍过两种图的遍历方法，深度优先和广度优先。搜索引擎采用的是广度优先搜索策略。具体点讲的话，那就是，我们先找一些比较知名的网页（专业的叫法是权重比较高）的链接（比如新浪主页网址、腾讯主页网址等），作为种子网页链接，放入到队列中。爬虫按照广度优先的策略，不停地从队列中取出链接，然后取爬取对应的网页，解析出网页里包含的其他网页链接，再将解析出来的链接添加到队列中。

基本的原理就是这么简单。但落实到实现层面，还有很多技术细节。我下面借助搜集阶段涉及的几个重要文件，来给你解释一下搜集工程都有哪些关键技术细节。

1.待爬取网页链接文件：links.bin

在广度优先搜索爬取页面的过程中，爬虫会不停地解析页面链接，将其放到队列中。于是，队列中的链接就会越来越多，可能会多到内存放不下。所以，我们用一个存储在磁盘中的文件（links.bin）来作为广度优先搜索中的队列。爬虫从links.bin文件中，取出链接去爬取对应的页面。等爬取到网页之后，将解析出来的链接，直接存储到links.bin文件中。

这样用文件来存储网页链接的方式，还有其他好处。比如，支持断点续爬。也就是说，当机器断电之后，网页链接不会丢失；当机器重启之后，还可以从之前爬取到的位置继续爬取。

关于如何解析页面获取链接，我额外多说几句。我们可以把整个页面看作一个大的字符串，我们可以利用字符串匹配算法，在这个大字符串中，搜索这样一个网页标签，然后顺序读取之间的字符串。这其实就是网页链接。

2.网页判重文件：bloom_filter.bin

如何避免重复爬取相同的网页呢？这个问题我们在位图那一节已经讲过了。使用布隆过滤器，我们就可以快速并且非常节省内存地实现网页的判重。

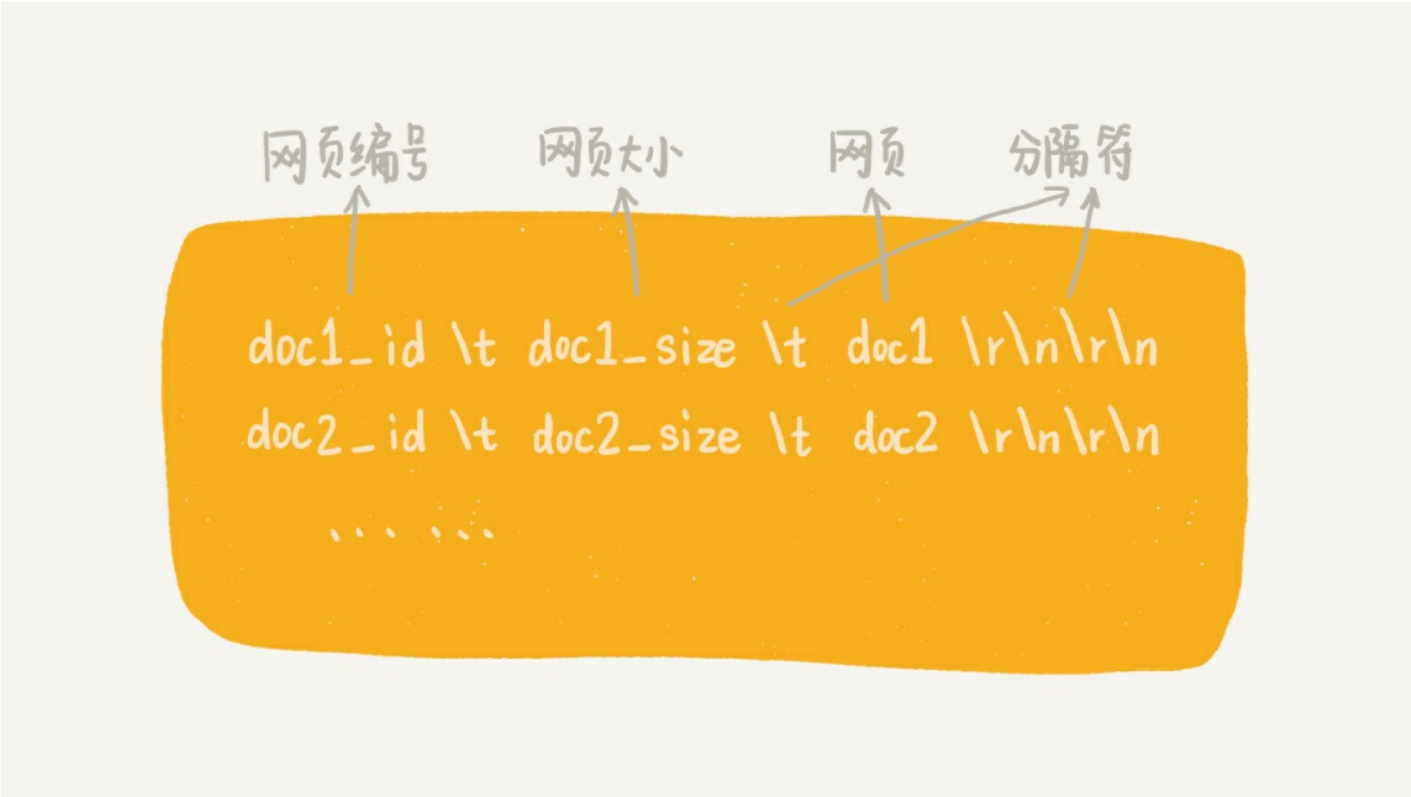
不过，还是刚刚那个问题，如果我们把布隆过滤器存储在内存中，那机器宕机重启之后，布隆过滤器就被清空了。这样就可能导致大量已经爬取的网页会被重复爬取。

这个问题该怎么解决呢？我们可以定期地（比如每隔半小时）将布隆过滤器持久化到磁盘中，存储在bloom_filter.bin文件中。这样，即便出现机器宕机，也只会丢失布隆过滤器中的部分数据。当机器重启之后，我们就可以重新读取磁盘中的bloom_filter.bin文件，将其恢复到内存中。

3.原始网页存储文件：doc_raw.bin

爬取到网页之后，我们需要将其存储下来，以备后面离线分析、索引之用。那如何存储海量的原始网页数据呢？

如果我们把每个网页都存储为一个独立的文件，那磁盘中的文件就会非常多，数量可能会有几千万，甚至上亿。常用的文件系统显然不适合存储如此多的文件。所以，我们可以把多个网页存储在一个文件中。每个网页之间，通过一定的标识进行分隔，方便后续读取。具体的存储格式，如下图所示。其中，doc_id这个字段是网页的编号，我们待会儿再解释。



当然，这样的一个文件也不能太大，因为文件系统对文件的大小也有一定的限制。所以，我们可以设置每个文件的大小不能超过一定的值（比如1GB）。随着越来越多的网页被添加到文件中，文件的大小就会越来越大，当超过1GB的时候，我们就创建一个新的文件，用来存储新爬取的网页。

假设一台机器的硬盘大小是100GB左右，一个网页的平均大小是64KB。那在一台机器上，我们可以存储100万到200万左右的网页。假设我们的机器的带宽是10MB，那下载100GB的网页，大约需要10000秒。也就是说，爬取100多万的网页，也就是只需要花费几小时的时间。

4.网页链接及其编号的对应文件：doc_id.bin

刚刚我们提到了网页编号这个概念，我现在解释一下。网页编号实际上就是给每个网页分配一个唯一的ID，方便我们后续对网页进行分析、索引。那如何给网页编号呢?

我们可以按照网页被爬取的先后顺序，从小到大依次编号。具体是这样做的：我们维护一个中心的计数器，每爬取到一个网页之后，就从计数器中拿一个号码，分配给这个网页，然后计数器加一。在存储网页的同时，我们将网页链接跟编号之间的对应关系，存储在另一个doc_id.bin文件中。

爬虫在爬取网页的过程中，涉及的四个重要的文件，我就介绍完了。其中，links.bin和bloom_filter.bin这两个文件是爬虫自身所用的。另外的两个（doc_raw.bin、doc_id.bin）是作为搜集阶段的成果，供后面的分析、索引、查询用的。

分析

网页爬取下来之后，我们需要对网页进行离线分析。分析阶段主要包括两个步骤，第一个是抽取网页文本信息，第二个是分词并创建临时索引。我们逐一来讲解。

1.抽取网页文本信息

网页是半结构化数据，里面夹杂着各种标签、JavaScript代码、CSS样式。对于搜索引擎来说，它只关心网页中的文本信息，也就是，网页显示在浏览器中时，能被用户肉眼看到的那部分信息。我们如何从半结构化的网页中，抽取出搜索引擎关系的文本信息呢?

我们之所以把网页叫作半结构化数据，是因为它本身是按照一定的规则来书写的。这个规则就是**HTML语法规范**。我们依靠HTML标签来抽取网页中的文本信息。这个抽取的过程，大体可以分为两步。

第一步是去掉JavaScript代码、CSS格式以及下拉框中的内容（因为下拉框在用户不操作的情况下，也是看不到的）。也就是`<style></style>`，`<script></script>`，`<option></option>`这三组标签之间的内容。我们可以利用AC自动机这种多模式串匹配算法，在网页这个大字符串中，一次性查找`<style>`, `<script>`, `<option>`这三个关键词。当找到某个关键词出现的位置之后，我们只需要依次往后遍历，直到对应结束标签（`</style>`, `</script>`, `</option`）为止。而这期间遍历到的字符串连带着标签就应该从网页中删除。

第二步是去掉所有HTML标签。这一步也是通过字符串匹配算法来实现的。过程跟第一步类似，我就不重复讲了。

2.分词并创建临时索引

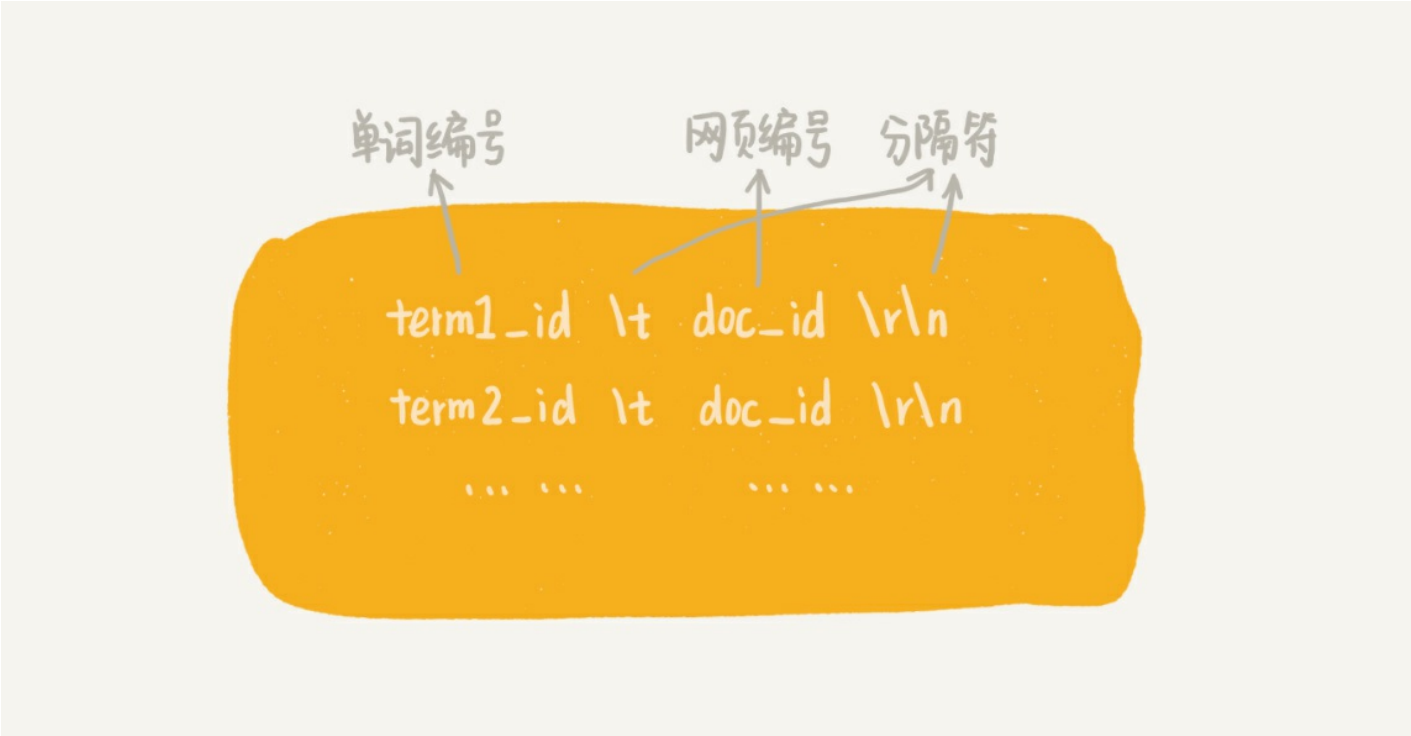
经过上面的处理之后，我们就从网页中抽取出了我们关心的文本信息。接下来，我们要对文本信息进行分词，并且创建临时索引。

对于英文网页来说，分词非常简单。我们只需要通过空格、标点符号等分隔符，将每个单词分割开来就可以了。但是，对于中文来说，分词就复杂太多了。我这里介绍一种比较简单的思路，基于字典和规则的分词方法。

其中，字典也叫词库，里面包含大量常用的词语（我们可以直接从网上下载别人整理好的）。我们借助词库并采用最长匹配规则，来对文本进行分词。所谓最长匹配，也就是匹配尽可能长的词语。我举个例子解释一下。

比如要分词的文本是“中国人民解放了”，我们词库中有“中国”“中国人”“中国人民”“中国人民解放军”这几个词，那我们就取最长匹配，也就是“中国人民”划为一个词，而不是把“中国”、“中国人“划为一个词。具体到实现层面，我们可以将词库中的单词，构建成Trie树结构，然后拿网页文本在Trie树中匹配。

每个网页的文本信息在分词完成之后，我们都得到一组单词列表。我们把单词与网页之间的对应关系，写入到一个临时索引文件中（tmp_Index.bin），这个临时索引文件用来构建倒排索引文件。临时索引文件的格式如下：



在临时索引文件中，我们存储的是单词编号，也就是图中的term_id，而非单词本身。这样做的目的主要是为了节省存储的空间。那这些单词的编号是怎么来的呢？

给单词编号的方式，跟给网页编号类似。我们维护一个计数器，每当从网页文本信息中分割出一个新的单词的时候，我们就从计数器中取一个编号，分配给它，然后计数器加一。

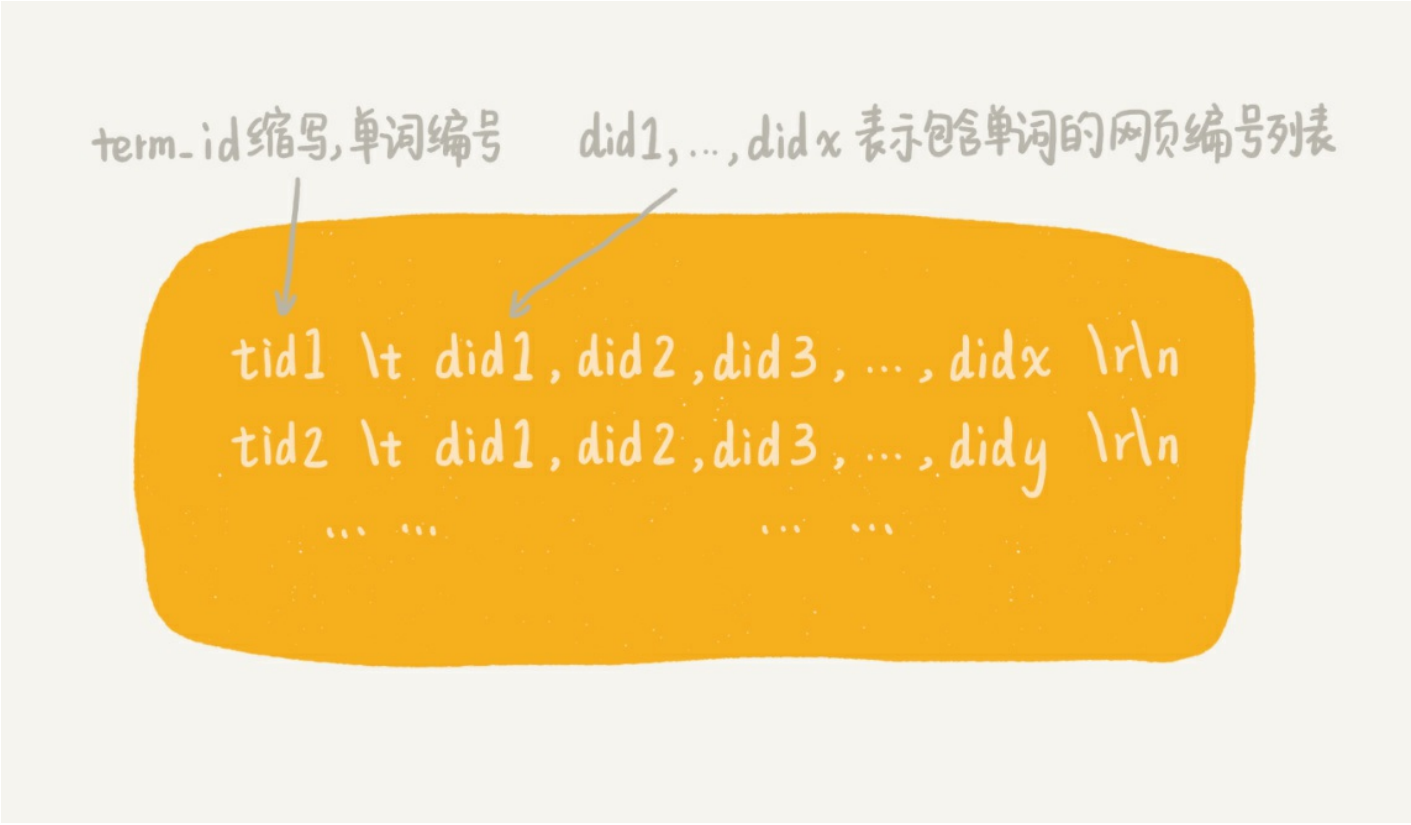
在这个过程中，我们还需要使用散列表，记录已经编过号的单词。在对网页文本信息分词的过程中，我们拿分割出来的单词，先到散列表中查找，如果找到，那就直接使用已有的编号；如果没有找到，我们再去计数器中拿号码，并且将这个新单词以及编号添加到散列表中。

当所有的网页处理（分词及写入临时索引）完成之后，我们再将这个单词跟编号之间的对应关系，写入到磁盘文件中，并命名为term_id.bin。

经过分析阶段，我们得到了两个重要的文件。它们分别是临时索引文件（tmp_index.bin）和单词编号文件（term_id.bin）。

索引

索引阶段主要负责将分析阶段产生的临时索引，构建成倒排索引。倒排索引（ Inverted index）中记录了每个单词以及包含它的网页列表。文字描述比较难理解，我画了一张倒排索引的结构图，你一看就明白。



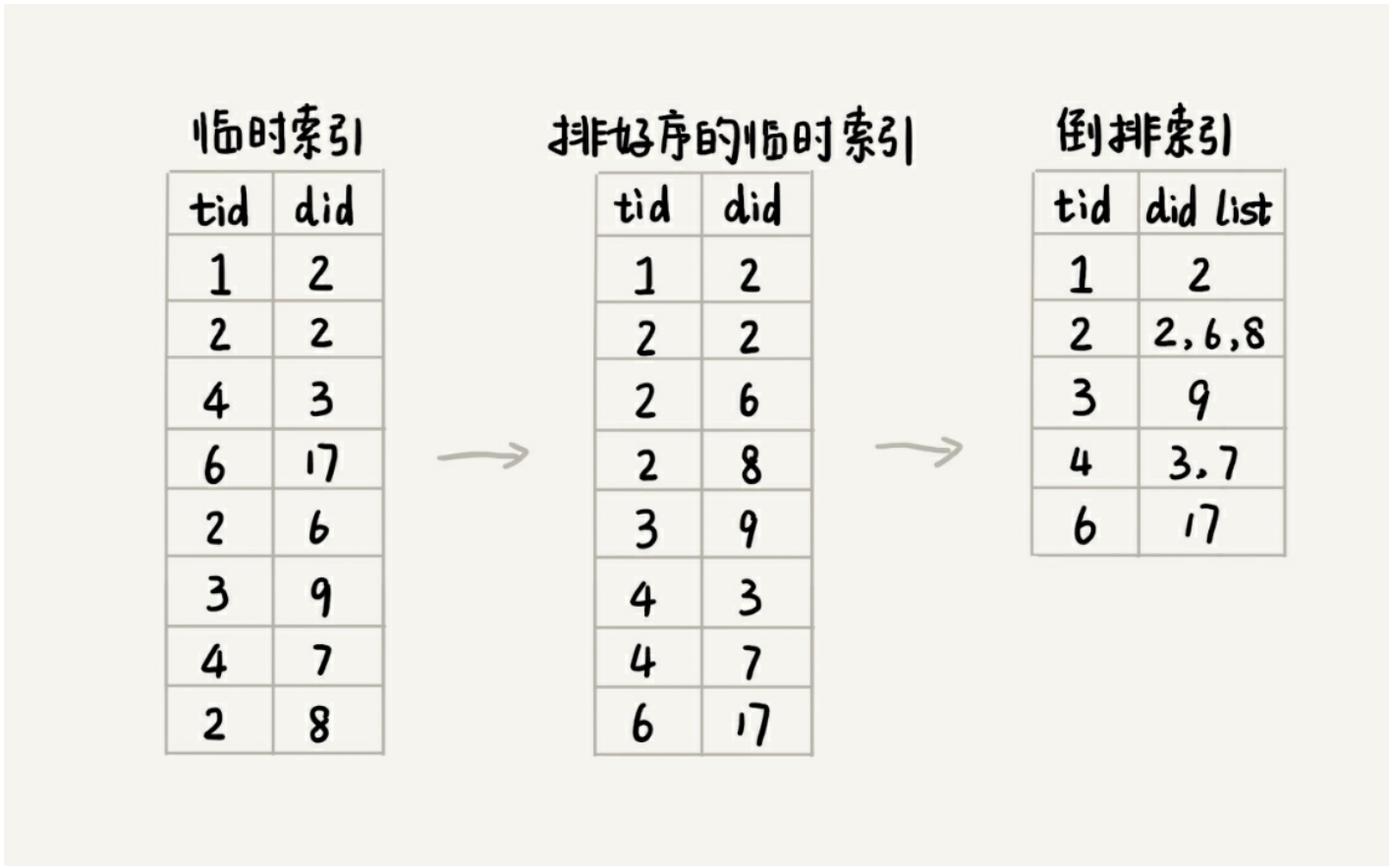
我们刚刚讲到，在临时索引文件中，记录的是单词跟每个包含它的文档之间的对应关系。那如何通过临时索引文件，构建出倒排索引文件呢？这是一个非常典型的算法问题，你可以先自己思考一下，再看我下面的讲解。

解决这个问题的方法有很多。考虑到临时索引文件很大，无法一次性加载到内存中，搜索引擎一般会选择使用**多路归并排序**的方法来实现。

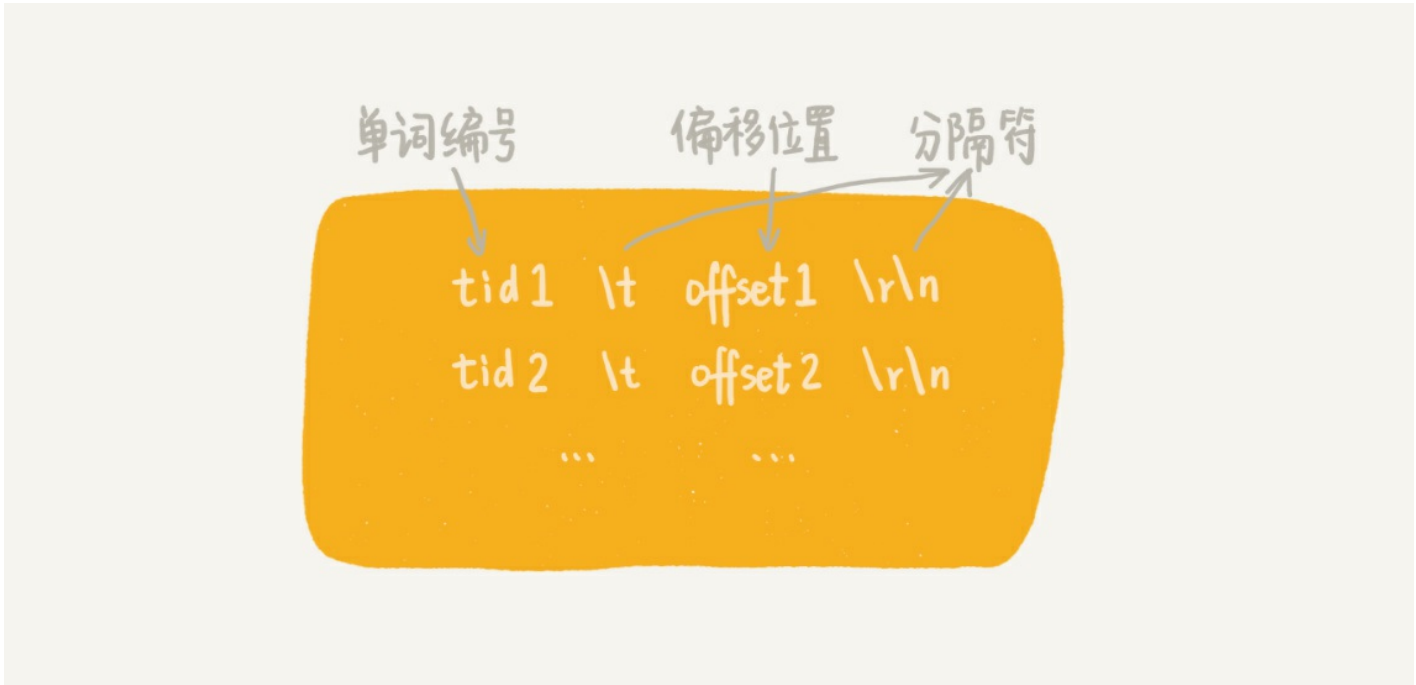
我们先对临时索引文件，按照单词编号的大小进行排序。因为临时索引很大，所以一般基于内存的排序算法就没法处理这个问题了。我们可以用之前讲到的归并排序的处理思想，将其分割成多个小文件，先对每个小文件独立排序，最后再合并在一起。当然，实际的软件开发中，我们其实可以直接利用MapReduce来处理。

临时索引文件排序完成之后，相同的单词就被排列到了一起。我们只需要顺序地遍历排好序的临时索引文件，就能将每个单词

对应的网页编号列表找出来，然后把它们存储在倒排索引文件中。具体的处理过程，我画成了一张图。通过图，你应该更容易理解。



除了倒排文件之外，我们还需要一个文件，来记录每个单词编号在倒排索引文件中的偏移位置。我们把这个文件命名为term_offset.bin。这个文件的作用是，帮助我们快速地查找某个单词编号在倒排索引中存储的位置，进而快速地从倒排索引中读取单词编号对应的网页编号列表。



经过索引阶段的处理，我们得到了两个有价值的文件，它们分别是倒排索引文件（index.bin）和记录单词编号在索引文件中

的偏移位置的文件（term_offset.bin）。

查询

前面三个阶段的处理，只是为了最后的查询做铺垫。因此，现在我们就要利用之前产生的几个文件，来实现最终的用户搜索功能。

- doc_id.bin：记录网页链接和编号之间的对应关系。
- term_id.bin：记录单词和编号之间的对应关系。
- index.bin：倒排索引文件，记录每个单词编号以及对应包含它的网页编号列表。
- term_offsert.bin：记录每个单词编号在倒排索引文件中的偏移位置。

这四个文件中，除了倒排索引文件（index.bin）比较大之外，其他的都比较小。为了方便快速查找数据，我们将其他三个文件都加载到内存中，并且组织成散列表这种数据结构。

当用户在搜索框中，输入某个查询文本的时候，我们先对用户输入的文本进行分词处理。假设分分词之后，我们得到k个单词。

我们拿这k个单词，去term_id.bin对应的散列表中，查找对应的单词编号。经过这个查询之后，我们得到了这k个单词对应的单词编号。

我们拿这k个单词编号，去term_offset.bin对应的散列表中，查找每个单词编号在倒排索引文件中的偏移位置。经过这个查询之后，我们得到了k个偏移位置。

我们拿这k个偏移位置，去倒排索引（index.bin）中，查找k个单词对应的包含它的网页编号列表。经过这一步查询之后，我们得到了k个网页编号列表。

我们针对这k个网页编号列表，统计每个网页编号出现的次数。具体到实现层面，我们可以借助散列表来进行统计。统计得到的结果，我们按照出现次数的多少，从小到大排序。出现次数越多，说明包含越多的用户查询单词（用户输入的搜索文本，经过分词之后的单词）。

经过这一系列查询，我们就得到了一组排好序的网页编号。我们拿着网页编号，去doc_id.bin文件中查找对应的网页链接，分页显示给用户就可以了。

总结引申

今天，我给你展示了一个小型搜索引擎的设计思路。这只是一个搜索引擎设计的基本原理，有很多优化、细节我们并未涉及，比如计算网页权重的PageRank算法、计算查询结果排名的tf-idf模型等等。

在讲解的过程中，我们涉及的数据结构和算法有：图、散列表、Trie树、布隆过滤器、单模式字符串匹配算法、AC自动机、广度优先遍历、归并排序等。如果对其中哪些内容不清楚，你可以回到对应的章节进行复习。

最后，如果有时间的话，我强烈建议你，按照我的思路，自己写代码实现一个简单的搜索引擎。这样写出来的，即便只是一个demo，但对于你深入理解数据结构和算法，也是很有帮助的。

课后思考

1. 图的遍历方法有两种，深度优先和广度优先。我们讲到，搜索引擎中的爬虫是通过广度优先策略来爬取网页的。搜索引擎为什么选择广度优先策略，而不是深度优先策略呢?

2. 大部分搜索引擎在结果显示的时候，都支持摘要信息和网页快照。实际上，你只需要对我今天讲的设计思路，稍加改造，就可以支持这两项功能。你知道如何改造吗?

欢迎留言和我分享，也欢迎点击“请朋友读”，把今天的内容分享给你的好友，和他一起讨论、学习。

精选留言

wei

思考题 1:

因为搜索引擎要优先爬取权重较高的页面，离种子网页越近，较大可能权重更高，广度优先更合适。

思考题 2:

摘要信息：
增加 summary.bin 和 summary_offset.bin。在抽取网页文本信息后，取出前 80-160 个字作为摘要，写入到 summary.bin，并将偏移位置写入到 summary_offset.bin。
summary.bin 格式：
doc_id \t summary_size \t summary \r\n\r\n
summary_offset.bin 格式：
doc_id \t offset \r\n
Google 搜索结果中显示的摘要是搜索词附近的文本。如果要实现这种效果，可以保存全部网页文本，构建搜索结果时，在网页文本中查找搜索词位置，截取搜索词附近文本。

网页快照：
可以把 doc_raw.bin 当作快照，增加 doc_raw_offset.bin 记录 doc_id 在 doc_raw.bin 中的偏移位置。
doc_raw_offset.bin 格式：
doc_id \t offset \r\n

2019-01-28 02:07

纯洁的憎恶

搜集：将广度优先搜索的优先队列存储在磁盘文件links.bin（如何解析网页内的链接？），有布隆过滤器判重并定期写入磁盘文件bloom_filter.bin，将访问到的原始网页数据存入磁盘文件doc_raw.bin，计数分配网页编号并与其链接对应关系存入磁盘文件doc_id.bin。

分析：首先抽取网页文本信息，依据HTML语法规范，通过AC自动机多模式串匹配算法，去除网页中格式化部分，提取文本内容。然后分词并创建临时索引，分词的目的是找到能够标识网页文本“身份”的特征，可借助词库（通过Trie树实现）搜索文本中与词库匹配的最长词语，因为一般情况下越长信息越多，越剧有表征能力（为什么英文简单？）。分词完成后得到一组用于表征网页的单词列表，与其对应的网页编号存入磁盘文件tmp_index.bin作为临时索引，为节省空间单词是以单词编号的形式写入，单词文本与编号的对应关系写入磁盘文本term_id.bin。

索引：通过临时索引构建倒排索引文件index.bin。倒排索引其实是以单词为主键，将临时索引中的多个相同单词行合并为一行。通过以单词为主键的排序算法，可以将相同单词的行连续排列在一起，之后只要将单词相同的连续行合并为一行即可。由于数据量大，应采用分治策略。最后建立所有单词在倒排索引文件中位置的索引文件term_offset.bin，以方便快速查找。

查询：先对搜索条件文本做分词处理，然后去term_id.bin查单词们的编号，再查term_offset.bin找到单词们在倒排索引中的位置，到index.bin找到每个单词对应的网页编号，通过网页出现次数、预评权重和统计算法（如pagerank、tf-idf）计算网页的优先次序并输出。最后在doc_in.bin中找到网页链接按序输出显示给用户。

这样理解对不?

2019-01-28 23:25

天凉好个秋

倒排索引中记录了每个单词以及包含它的网页列表，想问一下“倒排索引”这个名字是怎么来的？其中的“倒排”体现在哪里呢?

2019-01-28 10:56

作者回复

正排-》文档包含哪些单词

倒排-》单词被哪些文档包含

2019-01-29 11:22

alic

有没有代码实现的例子?

2019-01-28 10:13

作者回复

木有。等我有空了可以写下分享出来。

2019-01-29 11:23

Jerry银银

经过深入研究了一把，第一题终于有了比较清晰的答案：

从时间复杂度这个维度来考虑，BFS和DFS爬取互联网上所有的内容所需的时间是一样的。但是，我们设计爬虫系统的时候，不可能想着一次性爬完所有的网页，因为「量」太大了。所以，必须有一个优先级，不难想到：每一个网站的首页优先级最高，所以，我们肯定要先爬取每个网站的首页。从这一点出发，我们肯定要选取BFS。

但是，这里还有另外一个问题：如果我们爬完一个网站的首页之后，再爬取另外一个网站的首页，每次和不同网站服务器都要建立网络连接(TCP三次握手、HTTPs网站还要建立SSL握手等）都要花费大量的时间。如果总是按照BFS的策略来爬取，这中间花费的时间成本又太大了。所以，我想，中间肯定也是需要用DFS的。

我想到，可以使用一个优先级队列来维护需要爬取的网页。剩下的问题就是：该如何评估所需要爬取的网页的优先级呢？ 这个问题想了很久，依然不知道该如何计算机网页的优先级，难道这里也用PageRank类似的算法?

2019-01-30 08:31

往事随风，顺其自然

可以讲讲到排序索引和普通索引区别?

2019-01-29 21:34

Kudo

原理是看懂了，实现起来肯定会遇到各种各样的问题，手动实现一遍是有必要的，如果老师能提供一个参考代码就更好了。

2019-01-29 14:20

miss

问题1， 爬取网页时，如果采用深度优先算法，很有可能导致，栈溢出的现象把，所以一般不用深度优先算法

2019-01-29 12:53

yann [扬] :曹同学

带宽那里貌似有点问题，应该是100的

2019-01-29 09:32

王肖武

思考题1:深度优化借助栈这种数据结构，网页的深度是不可预测的，如果很深，栈大小会很大，内存可能会爆掉。

2019-01-29 08:51

纯洁的憎恶

思考题1。网络是动态变化的，所以爬虫的任务是在有限的时间里，爬到尽可能多的最重要的网页，而重要的网页一般是首页及其直接链接的网页。这样从首页开始一层层遍历更满足要求，尤其是在资源紧张的时候。所以BFS更适合。起码在大的调度次序上是这样。但在工程上要考虑网络通讯中“握手”次数过于频繁的问题，调度算法的具体实现细节会吸取DFS的优点。

2019-01-28 23:24

猫头鹰爱拿铁

老师 像那些专业搜索的例如学术搜索是怎么去限定数据源的搜索范围呢 是不是人为的维护了特定数据源列表还是用了相关算法

2019-01-28 17:54

作者回复

维护特定数据源的。

2019-01-29 11:20

在路上

争哥，Disruptor的无锁并发、环形数组，可以讲讲吗

2019-01-28 17:27

作者回复

可以的。

2019-01-29 11:20

小美

王老师，字典使用最长匹配？那例子中的”中国“”中国人“不就无法匹配到了吗

2019-01-28 10:35

作者回复

在这个例子中是的。“中国人好样的”这个句子分词就可以匹配到“中国人”

2019-01-29 11:22

[?]小佳[?]

老师，如果网页的内容是程序代码，而代码里有html的相应标签，字符串匹配算法是不是还需要考虑这点。

2019-01-28 10:15

作者回复

也是按照html标签来处理的，没有区别的。

2019-01-29 11:24

『LHCY』

作者讲的基本和elasticsearch原理查不多，可见有了算法基础以后了解一些中间件原理会容易很多，我最开始看es原理时一脸懵逼。

2019-01-28 09:13

莫弹弹

思考题1

如果使用深度优先遍历，那相当于全站克隆工具了，一搜全是这个网站的结果，变得像该网站的站内工具了

2019-01-28 09:12