数据结构与算法（18）：倒排索引

一、倒排索引

倒排索引（inverted index），也常被称为反向索引、置入档案或反向档案，是一种索引方法，被用来存储在全文搜索下某个单词在一个文档或一组文档中存储位置的映射。它是文档检索系统中最常用的数据结构。

有两种不同的倒排索引形式：

- 一条记录的水平反向索引（或者反向档案索引）包含每个引用单词的文档的列表。
- 一个单词的水平反向索引（或者完全反向索引）又包含每个单词在一个文档中的位置。

后者的形式提供了更多的兼容性（比如短语搜索），但是需要更多的时间和空间来创建。

以英文为例，下面是要被索引的文本：

- T_0="it is what it is"
- T_1="what is it"
- T_2="it is a banana"

我们就能得到下面的反向文件索引：

- "a": {2}
- "banana": {2}
- "is": {0, 1, 2}
- "it": {0, 1, 2}
- "what": {0, 1}

检索的条件"what", "is" 和 "it" 将对应这个集合：$\{0, 1\} \cap \{0, 1, 2\} \cap \{0, 1, 2\} = \{0, 1$

对相同的文字，我们得到后面这些完全反向索引，有文档数量和当前查询的单词结果组成的的成对数据。 同样，文档数量和当前查询的单词结果都从零开始。所以，"banana": {(2, 3)} 就是说"banana"在第三个文档里 T_2，而且在第三个文档的位置是第四个单词(地址为 3)。

- "a": {(2, 2)}
- "banana": {(2, 3)}
- "is": {(0, 1), (0, 4), (1, 1), (2, 1)}
- "it": {(0, 0), (0, 3), (1, 2), (2, 0)}

- "what": {(0, 2), (1, 0)}

如果我们执行短语搜索"what is it" 我们得到这个短语的全部单词各自的结果所在文档为文档0和文档1。但是这个短语检索的连续的条件仅仅在文档1得到。

有了这个索引系统，搜索引擎可以很方便地响应用户的查询，比如用户输入查询词“banana”，搜索系统查找倒排索引，从中可以读出包含这个单词的文档，这些文档就是提供给用户的搜索结果，而利用单词频率信息、文档频率信息即可以对这些候选搜索结果进行排序，计算文档和查询的相似性，按照相似性得分由高到低排序输出，此即为搜索系统的部分内部流程。

二、单词词典

单词词典是倒排索引中非常重要的组成部分，它用来维护文档集合中出现过的所有单词的相关信息，同时用来记载某个单词对应的倒排列表在倒排文件中的位置信息。在支持搜索时，根据用户的查询词，去单词词典里查询，就能够获得相应的倒排列表，并以此作为后续排序的基础。

对于一个规模很大的文档集合来说，可能包含几十万甚至上百万的不同单词，能否快速定位某个单词，这直接影响搜索时的响应速度，所以需要高效的数据结构来对单词词典进行构建和查找，常用的数据结构包括哈希加链表结构和树形词典结构。

2.1 哈希加链表

这种词典结构主要由两个部分构成：

主体部分是哈希每个哈希表项保存一个指针，指针指向冲突链表，在冲突链表里，相同哈希值的单词形成链表结构。之所以会有冲突链表，是因为两个不同单词获得相同的哈希值，如果是这样，在哈希方法里被称做是一次冲突，可以将相同哈希值的单词存储在链表里，以供后续查找。

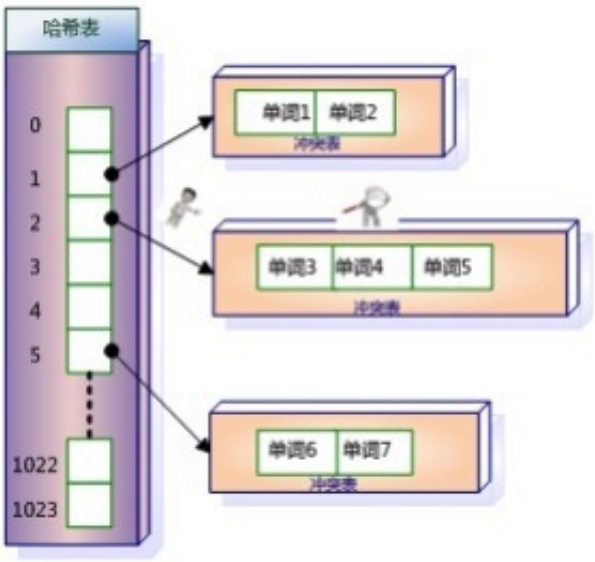

图 1-7　哈希加链表词典结构

在建立索引的过程中，词典结构也会相应地被构建出来。比如在解析一个新文档的时候，对于某个在文档中出现的单词T，首先利用哈希函数获得其哈希值，之后根据哈希值对应的哈希表项读取其中保存的指针，就找到了对应的冲突链表。如果冲突链表里已经存在这个单词，说明单词在之前解析的文档里已经出现过。如果在冲突链表里没有发现这个单词，说明该单词是首次碰到，则将其加入冲突链表里。通过这种方式，当文档集合内所有文档解析完毕时，相应的词典结构也就建立起来了。

在响应用户查询请求时，其过程与建立词典类似，不同点在于即使词典里没出现过某个单词，也不会添加到词典内。以图1-7为例，假设用户输入的查询请求为单词3，对这个单词进行哈希，定位到哈希表内的2号槽，从其保留的指针可以获得冲突链表，依次将单词3和冲突链表内的单词比较，发现单词3在冲突链表内，于是找到这个单词，之后可以读出这个单词对应的倒排列表来进行后续的工作，如果没有找到这个单词，说明文档集合内没有任何文档包含单词，则搜索结果为空。

2.2 树形结构

B树（或者B+树）是另外一种高效查找结构，图1-8是一个 B树结构示意图。B树与哈希方式查找不同，需要字典项能够按照大小排序（数字或者字符序），而哈希方式则无须数据满足此项要求。

B树形成了层级查找结构，中间节点用于指出一定顺序范围的词典项目存储在哪个子树中，起到根据词典项比较大小进行导航的作用，最底层的叶子节点存储单词的地址信息，根据这个地址就可以提取出单词字符串。