**索引的构建方法**

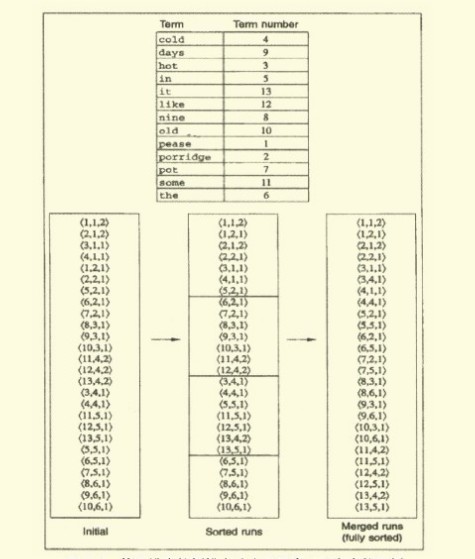
    根据信息检索导论（Christtopher D.Manning等著，王斌译）一书给的提示，我们可以选择两种构建索引的算法：BSBI算法，与SPIMI算法。

**BSBI算法，基于磁盘的外部排序算法**，此算法首先将词项映射成其ID的数据结构，如Hash映射。而后将文档解析成词项ID-文档ID对，并在内存中一直处理，直到累积至放满一个固定大小的块空间为止，我们选择合适的块大小，使之能方便加载到内存中并允许在内存中快速排序。  
    快速排序后的块转换成倒排索引格式后写入磁盘，建立倒排索引的步骤如下：

1. 对词项ID-文档ID进行排序
2. 将具有同一词项ID的所有文档ID放到倒排记录表中，其中每条倒排记录仅仅是一个文档ID
3. 将基于块的倒排索引写到磁盘上。

此算法假如说最后可能会产生10个块。  
（基于块的排序索引算法，改算法将每个块的倒排索引文件存入文件f1,...,fn中，最后合并成fmerged  
 如果该算法应用最后一步产生了10个块，那么接下来便会将10个块索引同时合并成一个索引文件。）

    合并时，同时打开所有块对应的文件，内存中维护了为10个块准备的读缓冲区和一个为最终合并索引准备的写缓冲区。每次迭代中，利用优先级队列（如堆结构或类似的数据结构）选择最小的未处理的词项ID进行处理。如下图所示（引自深入搜索引擎--海里信息的压缩、索引和查询，梁斌译）：



    读入该词项的倒排记录表并合并，合并结果写回磁盘中。需要时，再次从文件中读入数据到每个读缓冲区（基于磁盘的外部排序算法的更多可以参考：程序员编程艺术第十章、如何给10^7个数据量的磁盘文件排序）。

    BSBI算法主要的时间消耗在排序上，选择什么排序方法呢，简单的快速排序足矣，其时间复杂度为O（N\*logN），其中N是所需要排序的项（词项ID-文档ID对）的数目的上界。

**SPIMI算法，内存式单遍扫描索引算法**  
    SPIMI使用词项而不是其ID，它将每个块的词典写入磁盘，对于写一块则重新采用新的词典，只要硬盘空间足够大，它能索引任何大小的文档集  
    倒排索引 = 词典（关键词或词项+词项频率）+倒排记录表。建倒排索引的步骤如下：

1. 从头开始扫描每一个词项-文档ID（信息）对，遇一词，构建索引；
2. 继续扫描，若遇一新词，则再建一新索引块（加入词典，通过Hash表实现，同时，建一新的倒排记录表）；若遇一旧词，则找到其倒排记录表的位置
3. 在内存内基于分块完成排序，后合并分块；
4. 写入磁盘。

其伪码如下：  
**SPIMI与BSBI的主要区别**：  
    SPIMI，与BSBI算法一开始就整理出所有的词项ID-文档ID，并对它们进行排序的做法不同（而这恰恰是BSBI的做法），这里的每个倒排记录表都是动态增长的（也就是说，倒排记录表的大小会不断调整），同时，扫描一遍就可以实现全体倒排记录表的收集。  
    SPIMI这样做有两点好处:

1. 由于不需要排序操作，因此处理的速度更快，
2. 由于保留了倒排记录表对词项的归属关系，因此能节省内存，词项的ID也不需要保存。这样，每次单独的SPIMI-Invert调用能够处理的块大小可以非常大，整个倒排索引的构建过程也可以非常高效。

    但不得不提的是，由于事先并不知道每个词项的倒排记录表大小，算法一开始只能分配一个较小的倒排记录表空间，每次当该空间放满的时候，就会申请加倍的空间，  
与此同时，自然而然便会浪费一部分空间（当然，此前因为不保存词项ID，倒也省下一点空间，总体而言，算作是抵销了）。  
    不过，至少SPIMI所用的空间会比BSBI所用空间少。当内存耗尽后，包括词典和倒排记录表的块索引将被写到磁盘上，但在此之前，为使倒排记录表按照词典顺序来加快最后的合并操作，所以要对词项进行排序操作。

**小数据量与大数据量的区别**

    在小数据量时，有足够的内存保证该创建过程可以一次完成；  
    数据规模增大后，可以采用分组索引，然后再归并索 引的策略。该策略是，

1. 建立索引的模块根据当时运行系统所在的计算机的内存大小，将索引分为 k 组，使得每组运算所需内存都小于系统能够提供的最大使用内存的大小。
2. 按照倒排索引的生成算法，生成 k 组倒排索引。
3. 然后将这 k 组索引归并，即将相同索引词对应的数据合并到一起，就得到了以索引词为主键的最终的倒排文件索引，即反向索引。