<https://blog.csdn.net/yankai0219/article/details/8185872>

**转载请注明出处**[**http://blog.csdn.net/yankai0219/article/details/8185872**](http://blog.csdn.net/yankai0219/article/details/8185872) **问题一：**

**百度面试题：（第一部分：Top K 算法详解）**

    搜索引擎会通过日志文件把用户每次检索使用的所有检索串都记录下来，每个查询串的长度为1-255字节。

    假设目前有一千万个记录（这些查询串的重复度比较高，虽然总数是1千万，但如果除去重复后，不超过3百万个。一个查询串的重复度越高，说明查询它的用户越多，也就是越热门。），请你统计最热门的10个查询串，要求使用的内存不能超过1G。

**分析：**

     1.

|  |
| --- |
| 1B(byte) = 8bit  1KB = 1024B  1MB = 1024KB  1GB = 1024MB  故1千万个长度为1~255字节的查询串的大小为 ：256\*4\*250W=1KB\*250W 1KB\*250W/1024=2441.4MB = 2.38G  规定的内存不能超过1G |

     2.根据原文v\_July\_v大牛的分析，

          第一步：对字符串进行Hash，这样可以将一千万个记录缩小为三百万个记录，这样就可以将记录写入1G内存中。

          第二步：对Hash后的记录，进行比较。

     3.针对第一步HASH，给出简要说明，详细程序我也会在文章末尾给出

          1）数据结构

               （1）hash表结构中桶的结构

|  |
| --- |
| **typedef** **struct** \_HASH\_TABLE{  **unsigned** **int** hit\_count; //代表与该HASH值相冲突的key的个数                   NODE \* keys\_chain ;  }HASH\_TABLE; |

               （2）Hash表的桶中结点的结构

|  |
| --- |
| **typedef** **struct** \_NODE{                   type key ;//保存key的原始数据，typedef char\* type;  **int** key\_len ;//key的长度  **unsigned** **int** occur\_count; //同一个key出现的次数，这个是关键，将用于第二部  **struct** \_NODE \*next ;  }NODE; |

          2）所使用函数

               （1）与Hash操作有关的函数

|  |
| --- |
| /\*hash table length is fixed (value is BUCKET\_LEN)\*/  HASH\_TABLE\* **yk\_hash\_table\_init**();    /\* A Simple Hash Function \*/  **unsigned** **int** **yk\_simple\_hash\_string** (**char** \*str, **int** str\_len);  NODE \* **yk\_find\_data\_from\_hash\_table**( HASH\_TABLE \* hashTb,type key,**int** key\_len);  STATUS **yk\_insert\_data\_into\_hash\_table**( HASH\_TABLE \* hashTb,type key,**int** key\_len);  **void**   **yk\_output\_result**( HASH\_TABLE \* hashTb); |

     （2）与字符串操作有关的函数

|  |
| --- |
| **void** **yk\_print\_str**( **char** \* str, **int** str\_len);  **void** **yk\_cpy\_str**( **char** \*dst, **char** \*src, **int** str\_len);  **int** **yk\_strcmp**( **char** \* str1, **char** \*str2, **int** len1, **int** len2); |

          3）简要说明

                    Hash表这一数据结构，刚入手时真心搞不明白，不知道是为何要发明这一数据结构，也搞不懂如何使用这一数据结构。通过先编写最基本的Hash的操作，大致明白了Hash如何操作和如何使用。之后，又尝试实现TOPK算法，进一步了解了Hash的用法。

                    其实Hash的用法很简单，无非就是Hash表的初始化+数据插入+数据处理。Hash表的初始化不仅仅是固定大小的数组，也可以是长度可变的。在Nginx的Hash操作中，就是大小可变的Hash表。这儿还是值得学习。

**程序说明：**

                    目前编写的程序，没有实现排序功能，还需要我继续完善，继续学习排序算法。当前的程序请到资源中下载，网址：

<http://download.csdn.net/detail/yankai0219/4768858>

**问题二：**

给定a、b两个文件，各存放50亿个url，每个url各占64字节，内存限制是4G，让你找出a、b文件共同的url？

解决方案：

方案1：可以估计每个文件安的大小为5G×64=320G，远远大于内存限制的4G。所以不可能将其完全加载到内存中处理。考虑采取分而治之的方法。

    遍历文件a，对每个url求取hash(url)%1000，然后根据所取得的值将url分别存储到1000个小文件（记为a0,a1,...,a999）中。这样每个小文件的大约为300M。

    遍历文件b，采取和a相同的方式将url分别存储到1000小文件（记为b0,b1,...,b999）。这样处理后，所有可能相同的url都在对应的小文件（a0vsb0,a1vsb1,...,a999vsb999）中，不对应的小文件不可能有相同的url。然后我们只要求出1000对小文件中相同的url即可。

    求每对小文件中相同的url时，可以把其中一个小文件的url存储到hash\_set中。然后遍历另一个小文件的每个url，看其是否在刚才构建的hash\_set中，如果是，那么就是共同的url，存到文件里面就可以了。

方案解释说明：

     1）url相同的话，其hash值肯定是相同的，那么肯定会放入对应的小文件中。比如说a中的http://www.baidu.com的hash值为105，那么b中的http://www.baidu.com的hash值也为105。因此所有可能相同的url都在对应的小文件中（a0-b0,a1-b1）中，不对应的小文件中不可能有相同的url。

     2）在这个解决方案中，遍历文件a，根据hash值得到1000个小文件。类比到常见的hash操作，1000个小文件就是1000个桶，也就是1000个链表，这是这儿将其更改为1000个文件。

程序说明：

参考文献

<http://blog.csdn.net/v_july_v/article/details/6256463>

<http://bbs.csdn.net/topics/370253735>