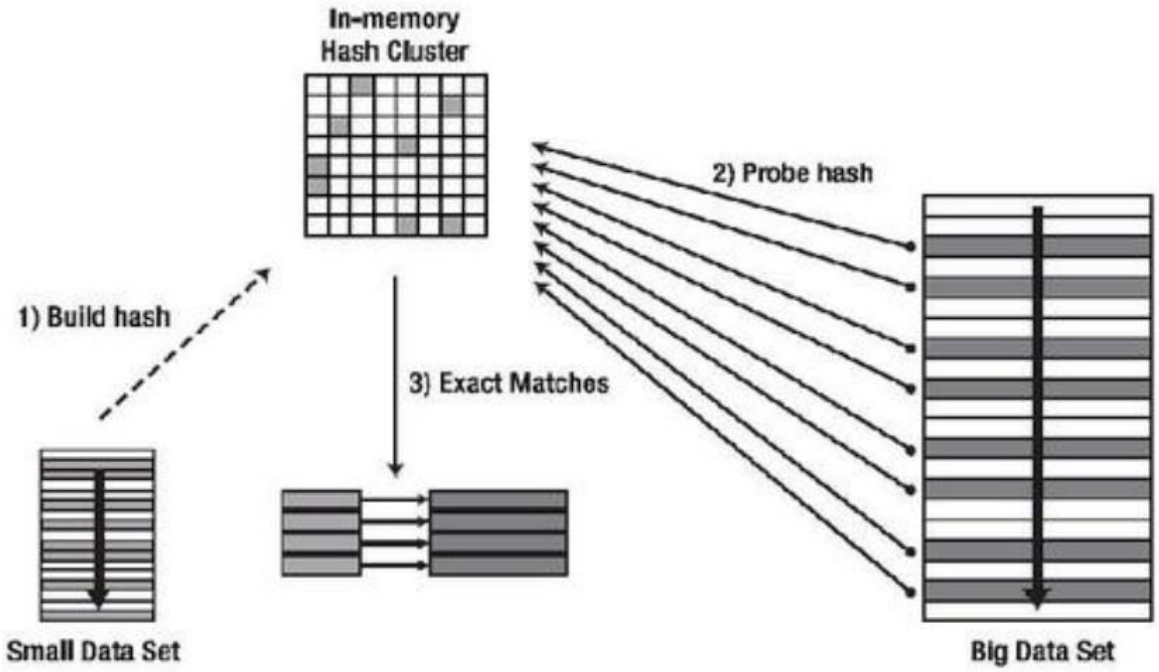
装 订 线



求解千万条短信查重问题

——侧重于哈希法求解

电子与信息工程学院 计算机类1班

1852409 李佳庚

装 订 线

1. 题目

有一千万条短信，有重复，以文本文件形式保存，一行一条，请找出重复出现最多的20条。

2. 问题的分析

对于这个问题，首先我们要对它进行分析。
题目中的下面几个词语尤其需要注意：

1. **“千万条”**
2. **“有重复”**
3. **“文本形式”**
4. **“一行一条”**

对于千万条这个关键词，它相当于是给出了短信的数据规模。要知道，正常一条短信大约60~70字左右，千万条意味着短信总字数很可能超过亿字。数据量是十分庞大的，于是算法的时间复杂度应该提到比较重要的位置进行考虑。

而有重复这个信息，告诉了我们这个问题存在的关键。并不是所有的短信都是不重复的。因为有了这个重复条件，所以我们才可以利用某些算法进行问题的解决。

文本形式这个词也是十分重要，它表示这我们可以使用ascii码对原本字符类型的信息进行转化，这一点，对无论是何种方法，都是十分重要的。

一行一条这个并没有太多意义。在我看来，只是方便写程序而已。一行一条标示着每条短信都以“\n”结尾，方便循环的书写而已。

由此，我们可以得知，我们需要一个能够进行文字匹配的、数据处理的、算法复杂度低的算法。

3. 诸多方法的提出

1. **直接暴力求解。**

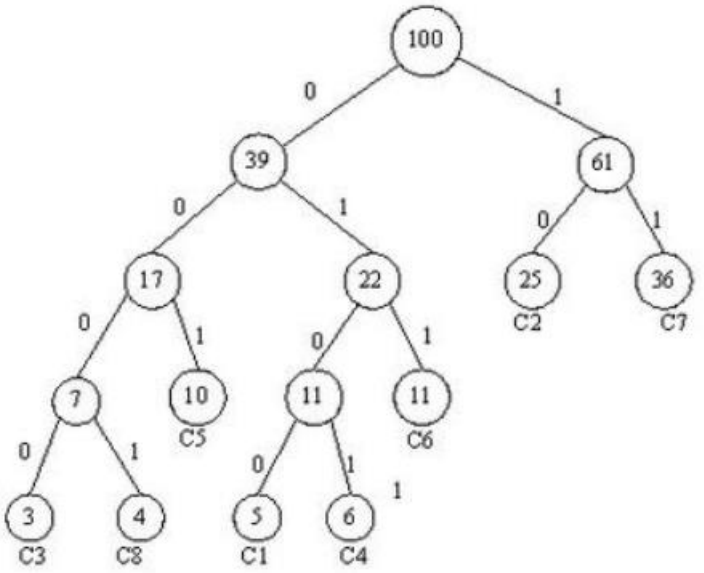
   不妨直接算。写个for循环，对第一个短信，要遍历后面9999999条短信，依次匹配。相同的就提出去，之后再也不看了，不相同的短信留在原地，继续进行匹配。

可以看出这样操作之后，得出最终答案的速度是很慢的。大致估算一下时间复杂度。这样对每一个短信都要匹配剩下所有短信的做法，应该是n^2的复杂度才对。而且这种算法没有利用到上面所说的ascii->int的便捷。

肯定不是一个好的方法。

2. **哈夫曼树编码。**

   其实很熟悉了，不久之前才写过。



   把所有的短信先进行哈夫曼编码，将其转化为01比特串。即使从这个时候开始进行匹配，不再进行下面的优化算法，那肯定还是比上面那个快的。毕竟用的是bit。

   在这之后，我们可以根据不同短信编码出的哈夫曼树依次进行权重值的比较。比如a短信根节点权重值为100，而b短信根节点权重值为10。那显然a和b不是同一条短信。首先长短肯定不同了。

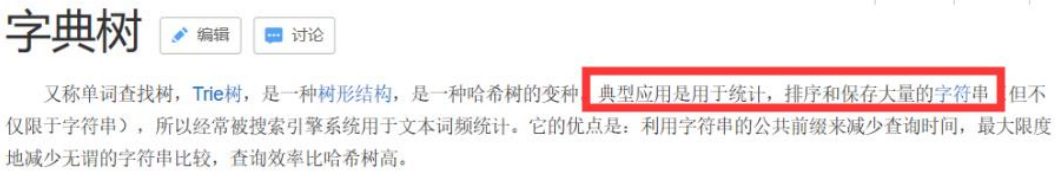
A短信有100个字符，而b短信只有10个。这样，我们比较一次int型的权重值，就代表着我们一下子比较了很多种类字符的总数。大大加快了比较的速度。如果我们的哈夫曼树是一个比较均衡的树，那么我们的匹配速率将更是大大提高。

可以看出哈夫曼编码的确优于之前的暴力查找。但是这并非是最简单的思路。

3. **字典树。**

字典树是专门被创造出来应付这类问题的数据结构。

百度百科上对它有如下说明


字典树 编辑 讨论

又称单词查找树，Trie树，是一种树形结构，是一种哈希树的变种 典型应用是用于统计，排序和保存大量的字符串 （但不仅限于字符串），所以经常被搜索引擎系统用于文本词频统计。它的优点是：利用字符串的公共前缀来减少查询时间，最大限度地减少无谓的字符串比较，查询效率比哈希树高。

具体的操作，课上的同学们已经教会我了。

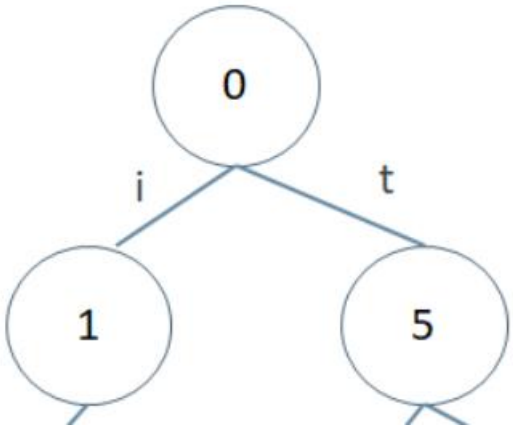
首先，我们对这千万条短信进行遍历。但是只看第一个字母。

可见字符127个，从1~128（如果有中文的话，那么还有负数）都有可能出现。

我们不妨建立一个多叉树。

这个树会根据在第一个字读到的数据自动从根节点创建叶子节点。

就比如：



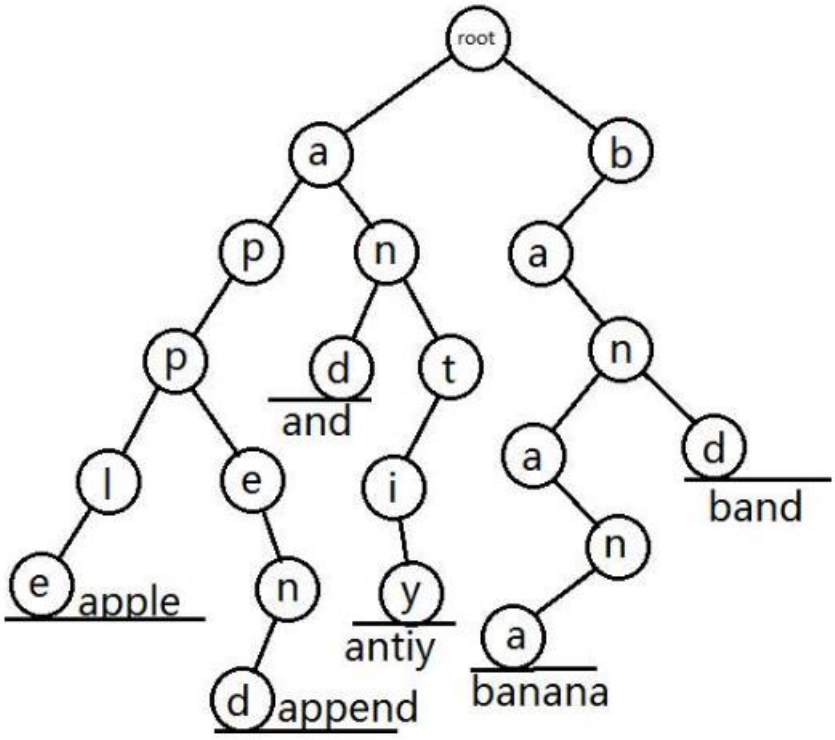
：

如上图，表示的是千万条短信的第一个字符，i出现了1个单位次，而t出现了5个单位次。也就是说，千万条短信的第一个字符只有i和t两种。

于是千万条短信被编写成从根节点开始的每层最多127个子节点，最多达短信字数最大限制层数的多叉树。

最后，在每个叶子节点中都有一个权重，对权重进行排序便可以得到重复最多的短信。

我找了网上的一个字典树，大概是长这个样子的。



.

这种方法可以说是比较完备了。仅仅遍历一次千万条短信的内容，甚至还不需要遍历完，就可以计算出最后的结果。

相较于前面两种方法，可以说是大大优化了。

但是还不是最好的。

4. 方法的局限性及最优方法的提出

上面那几个方法没有啥大的优点。

缺点就是慢。

在研讨课上，我觉得我想出来的那个方法应该是最快最有效的。

主要的思路就是hash链表->map->优先队列。

5. 具体思路

5.1 散列表

要说hash表。

那之前得说说哈希到底是个什么。

我也算是再复习一遍。

Hash函数我个人认为就是进行一个映射。在这道题里面，便是将任意大小的字符串进行从字符串到int型数据转换的函数。

这个函数是人为构造的，在构造的过程当中，尽量利用数学的方法，保证其是均匀的。Hash出来的结果或是完全不同的，或是等可能地分布在槽位中的。

对于字符串string而言，我现在暂时采用Java中hash函数对string类型的做法：乘法散列。

11.3.2 乘法散列法

构造散列函数的乘法散列法包含两个步骤。第一步，用关键字 k 乘上常数 $A(0<A<1)$，并提取 kA 的小数部分。第二步，用 m 乘以这个值，再向下取整。总之，散列函数为：

$$h(k)=\lfloor m(kA \bmod 1)\rfloor$$

这里“$kA \bmod 1$”是取 kA 的小数部分，即 $kA-\lfloor kA\rfloor$。

上图摘自clrs的第11章——散列表。

乘法散列的优点是不需要考虑太多。而且如果是字符串的话，可以做到：长的字符串散列值大，短的字符串散列值比较小。实现如此的特性。

从而不太可能发生hash碰撞。

如果短信相同，那么得到的hash值一定是相等的。我们完全可以在槽中添加新的节点来表示有更多的短信彼此相同，

具体类似于：

11.5 完全散列

使用散列技术通常是个好的选择，不仅是因为它有优异的平均情况性能，而且当关键字集合是**静态**(static)时，散列技术也能提供出色的最坏情况性能。所谓静态，就是指一旦各关键字存入表中，关键字集合就不再变化了。一些应用存在着天然的静态关键字集合，如程序设计语言中的保留字集合，或者 CD-ROM 上的文件名集合。一种散列方法称为**完全散列**(perfect hashing)，如果该方法进行查找时，能在最坏情况下用 $O(1)$次访存完成。

我们采用两级的散列方法来设计完全散列方案，在每级上都使用全域散列。图 11-6 描述了该方法。

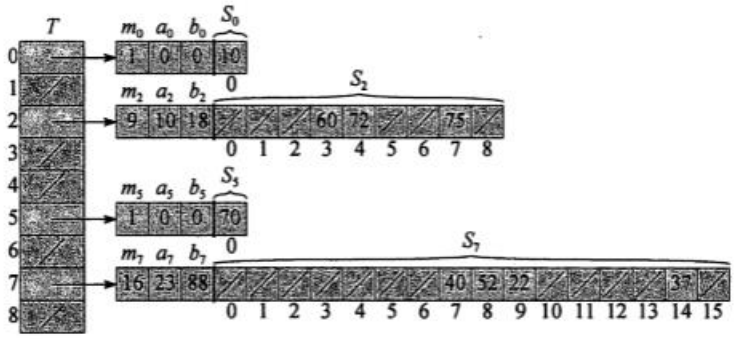


图 11-6 利用完全散列技术来存储关键字集合 $K=\{10, 22, 37, 40, 52, 60, 70, 72, 75\}$。外层的散列函数为 $h(k)=((ak+b) \bmod p) \bmod m$，这里 $a=3$，$b=42$，$p=101$，$m=9$。例如，$h(75)=2$，因此，关键字 75 散列到表 T 的槽 2 中。一个二级散列表 S_j 中存储了所有散列到槽 j 中的关键字。散列表 S_j 的大小为 $m_j=n_j^2$，并且相关的散列函数为 $h_j(k)=((a_jk+b_j) \bmod p) \bmod m_j$。因为 $h_2(75)=7$，故关键字 75 被存储在二级散列表 S_2 的槽 7 中。二级散列表没有冲突，因而查找操作在最坏情况下所需的时间为常数

（但是算法思想不同于完全散列）

5.2 c++ map

map是stl里面的一个容器。内部由红黑树实现，思想是构造一个有first key到second key的映射。然后对于第一个键值，可以根据他进行排序之类的操作。不如把map看作可以排序的pair。

class template

std::map <map>

```
template < class Key,                                     // map::key_type
           class T,                                       // map::mapped_type
           class Compare = less<Key>,                     // map::key_compare
           class Alloc = allocator<pair<const Key,T> >    // map::allocator_type
           > class map;
```

Map

Maps are associative containers that store elements formed by a combination of a *key value* and a *mapped value*, following a specific order.

In a `map`, the *key values* are generally used to sort and uniquely identify the elements, while the *mapped values* store the content associated to this *key*. The types of *key* and *mapped value* may differ, and are grouped together in member type `value_type`, which is a pair type combining both:

```
typedef pair<const Key, T> value_type;
```

Internally, the elements in a `map` are always sorted by its *key* following a specific *strict weak ordering* criterion indicated by its internal comparison object (of type `Compare`).

我在将千万条短信全部hash之后，就立刻将hash表中的权重值和编号进行map的构造。比如，就会有：

1->asdaf

100->weqwe

100000->qwrqwe

99->sd45aweqeq sda

……

之类的映射。

然后根据first key，利用上一次报告中提到的优先队列，很轻松就可以得出从第一个到第10个究竟都是谁。

装

6. 心得体会

我个人以为我的做法是比较快的。

首先计算hash值，即使是千万条短信，加和速度是很快的。以短信为单位，时间复杂度仅仅为n。

订

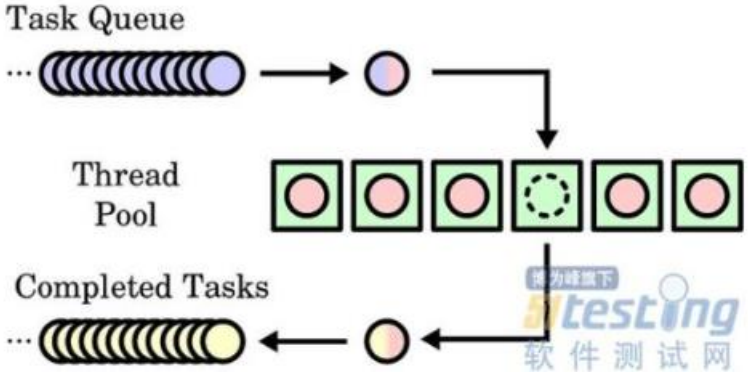
之后将hash table中的权重输入到map中，根据重复程度的大小，需要的时间不大相同。总之，时间复杂度也可算作为n。

利用 优先队列/堆排序 对map中的first key进行排序需要nlgn的时间。最后的话，大概是只需要nlgn的时间就可以完成整个千万条短信的统计匹配。

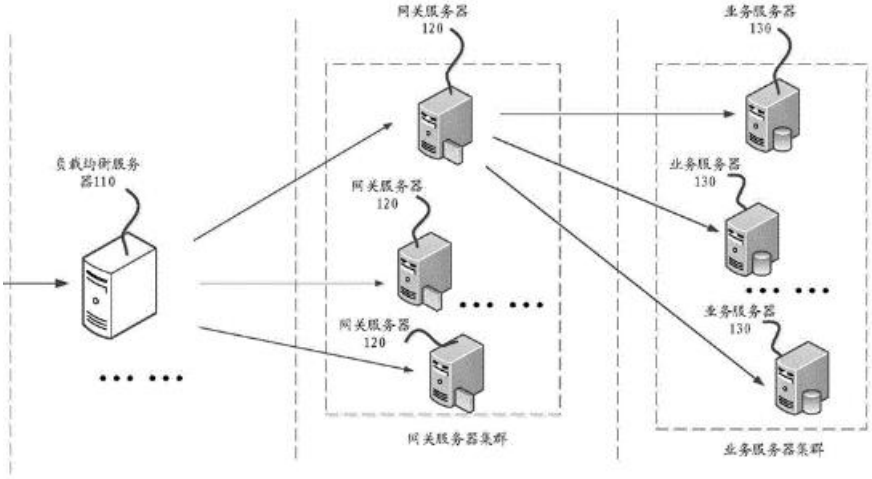
线

7. 除此之外其他的问题

其实对于这么大规模的数据，我们也不需要用一个线程，甚至一台电脑和它死磕。不如对千万条信息进行合理划分，利用多线程技术快速统计。



也不如构建分布式计算集群，以最简单的轮询法分配每一条短信给下属的机器。



总之，为了快速完成这个题目，方法还有很多很多。

8. 对本次研讨课的建议

1. 相比于上次研讨课，本次题目的难度有了很大的提高，我觉得很好。但是个人认为还是不够难。我希望看到的是我一下子完全想不出来，然后只有通过和同学们的不断研究探讨当中才能发现蛛丝马迹的题目。
   但是题目难度已经有了提高，很不错，继续努力!
2. 这次大家都很抓得住重点。我觉得还行。
3. 这些题目我得知是比较有来头的。我觉得真的可以。
4. 要硬是说不足的话，应该就是讨论的时间太长了。题目比较简单，也不需要那么长的时间谈论，应该用更多的时间来让下面的同学上去讲一讲自己对题目的理解和看法。
5. 祝研讨课越办越好。

电子与信息工程学院 计算机类 1 班 1852409 李佳庚
2019/11/17

装 订 线