**字符串匹配实验报告**

**数组实现array\_search：**

对于这个版本的编程实现，思路比较简单粗暴 就是通过建立两个数组来分别存放120w的与98w的字符串，再分别对其进行一一比较，时间花费在O(m\*n)级别，无论是时间开销还是空间开销都较大，是比较直接与初级的一种解法。

具体到实现来，我使用了指针数组的形式来存放字符串，数组内元素为一char型指针，每个char型指针则指向一个字符串。以此来进行字符串的存储与查找。

|  |
| --- |
| char \*\*p=NULL;  p = (char \*\*)malloc(N \* sizeof(char\*)); //地址数组  p[i] = (char \*)malloc(len \* sizeof(char)) //char指针指向一个字符串 |

将字符串都存进数组之后，则可建立循环来对其进行字符串的匹配。在本例中使用的是对所处两个文件中没两个字符串的每个字符char型进行一一比对。消耗十分巨大。 而这也从程序的输出结果中可以看出：

|  |
| --- |
| 70521 1122231076 986004 516816 |

消耗了70521kb的内存，最终要的是进行了11亿次的比较，效率十分低下。

**哈希表实现hashtable\_search:**

通过hash表的碰撞性来进行匹配，使用了BKDR哈希函数来将字符串转变为一unsigned int，因此字符串的匹配可以转变为该unsigned int数是否相同。但同时我们也得考虑hash碰撞的情况，因此使用拉链法来处理碰撞的情形，链表内元素结构为：

|  |
| --- |
| struct linklist {  unsigned int val;  struct linklist \*next;  }; |

对于hash值相同的元素放入一串链表内，再具体比较链表内元素的值来判断是否匹配。相比在字符内对char型变量一个个进行匹配，对字符串hash后的数字进行比较效率要高得多，只需对hash值相同的一串链表进行匹配而不是对整个目标字符串集进行匹配，理论上查找效率将提高m倍。（m为哈希表数组的长度）

从结果也可以看出：

|  |
| --- |
| 87091 8615167 986004 516818 |

比较次数为860万，相比于第一个暴力一个个匹配的实现方式下降了几个数量级。又由于hash函数再转换时的冲突特性，准确率相比与每个字符串都匹配的方式有所下降。

**Bloom Filter实现：**

布隆过滤器其空间效率与查询时间都远超一般的算法，属于在保证准确率的前提下以时间换空间的操作。其使用多个hash函数来相互独立地对一个字符串进行转换，在匹配时只有这多个hash函数转换后的值在比特向量中对应位置全为1时才认为匹配成功。为了降低错误率，就势必要选择多个hash函数，hash函数的个数k可有给定错误率后由公式计算得出。

本人使用char型数组来表示一比特向量，对单个比特的操作则在其对应的char[location]上进行。

|  |
| --- |
| char \*bfarray = (char \*)malloc((size) \* sizeof(char)); |

使用比特向量后存储空间可大为缩小，从结果上：

|  |
| --- |
| 35615 11057096 986004 516816 |

内存使用为35615kb，为3种实现方式中最小。又由于以时间换空间，再保证了准确率的前提下比较次数会比第二种实现方式要高。

**总结：**

Hash函数真好用，以准确率为代价将查找时间降低到O(1)级别，并通过了Bloom Filter来通过多个hash降低错误率，虽然牺牲了一些时间，但查找时间相比于传统的查找方法仍十分迅速。