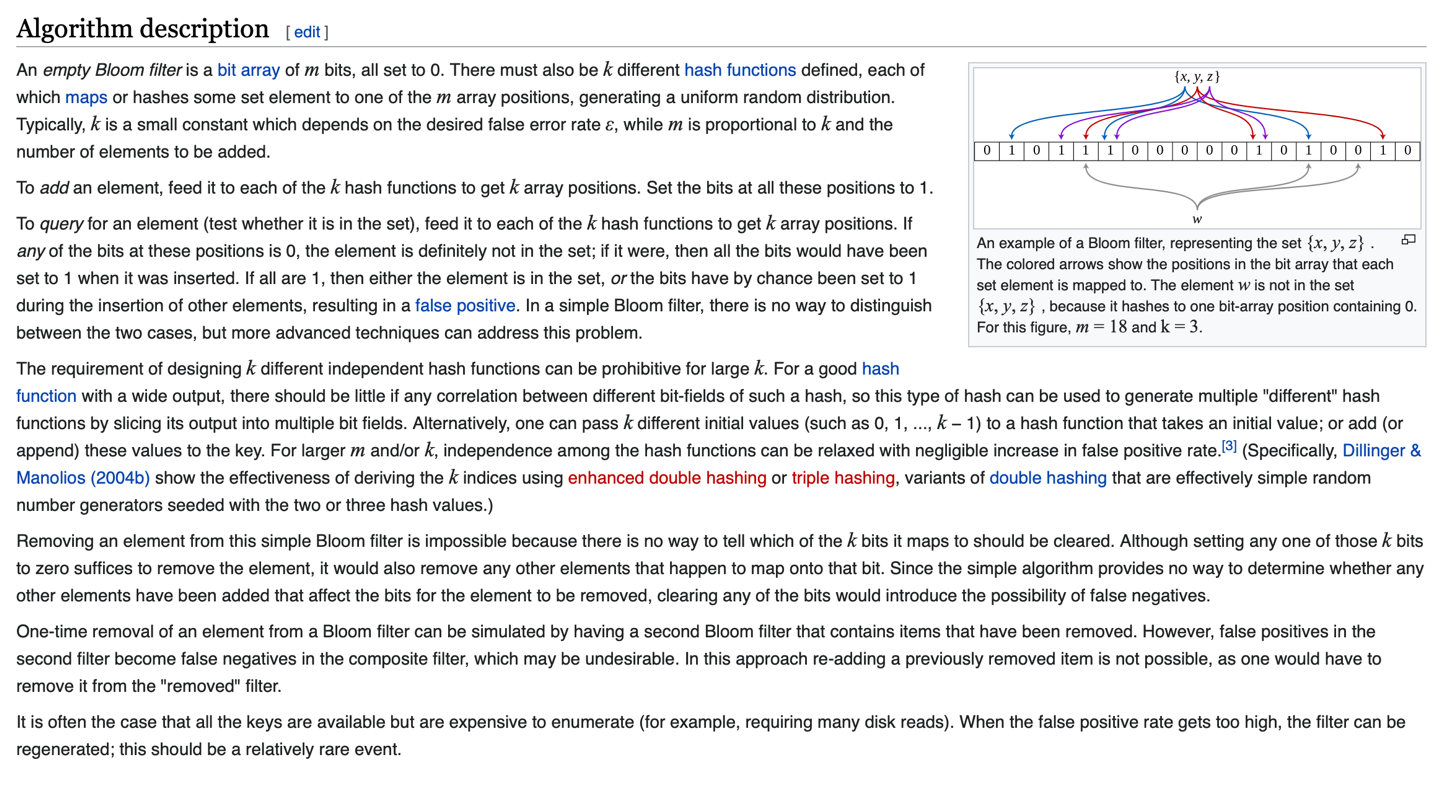
**第11周上机实验报告**

# 一、题目：

实现bloom filter



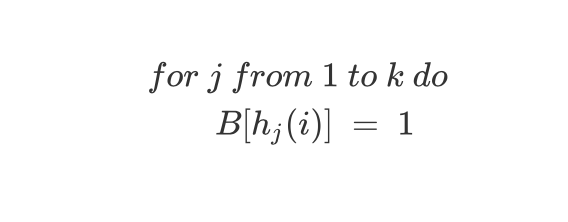
# 二、算法思路：

**BloomFilter**

* 包含一个bool类型数组和k个独立的哈希函数。
* 初始化时，需确定bool类型数组的大小，以及哈希函数的数量。
* 哈希函数是为了将字符串映射成为无符号整数，参考阅读材料，使用md5函数将字符串映射成为一个16字节无符号整数，取其中2字节为一个哈希值。
  + 最多可以取16/2=8个哈希值，因此哈希函数的数量不能大于8。
  + 同时哈希函数的哈希值由于是2字节无符号整数，其值的范围为2^(2\*8)=2^16次方。因此哈希表的大小m不能大于2^16。

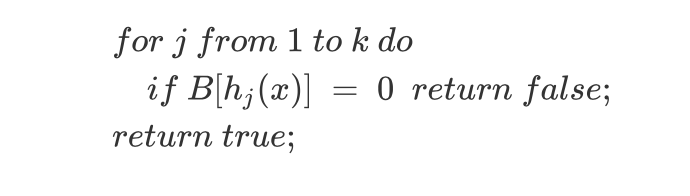
**Insert**

**伪代码**

****

**Query**

**伪代码**

****

# 三、程序设计框架：

## 核心类的名称及功能

**核心类Bloomfliter**

**成员变量**

1. //MD5 hash result,总是16bytes大小
2. **static** **const** **size\_t** MD5\_result\_size\_bytes = 16;
4. //bloomfilter storage in bits 2^16
5. **const** **size\_t** bloomfilter\_store\_size ;
7. **const** **size\_t** hash\_function\_count;
9. std::vector<**bool**>bloomfilter\_store\_;
10. **size\_t** object\_count\_;
11. unsigned **char**\* MD5\_hash\_result\_buffer;

* Md5\_result\_size\_bytes
  + Md5函数返回结果是16字节的
* Bloomfilter\_store\_size
  + 布隆过滤器的存储空间（哈希表）大小
* Hash\_function\_size
  + 哈希函数的个数
* Bloomfilter\_store\_
  + 布隆过滤器的存储内容，使用vector<bool>，是因为布隆过滤器需要对一个bit进行操作，bool变量理论上说只需要一个bit就可以存下。
  + 但是事实上c++中bool变量的大小是一个字节也就是（8个比特），鉴于本实验并不需要投入实际使用，在此用vector<bool>来象征对单个bit的操作，不影响正确性或是空间利用率的分析。
  + C++的库<bitset>可以做到对bit操作，但是由于该库的初始化必须声明所需空间大小，且无法修改，不方便更改哈希表的大小，因此探究哈希表大小和false positive会很不方便，遂没有采用。
* Object\_count\_
  + 统计一共插入了多少元素
* Md5\_hash\_result\_buffer
  + 存储Md5哈希函数的返回结果

**成员函数(只介绍核心函数)**

**哈希函数**

* 将输入的string利用MD5函数进行哈希，将结果储存在md5\_hash\_result\_buffer中
  + - 这里的md5函数使用的第三方库<openssl/md5>提供的实现。

1. **void** hash(**const** std::string& val)**const**
2. {
3. //强制类型转换，仅仅是复制了bits, string.data后无\0和.c\_string()相比
4. **const** unsigned **char**\* **const** md5\_input\_val = **reinterpret\_cast**<**const** unsigned **char**\*>(val.data());
5. **const** **size\_t** md5\_input\_length = val.length();
6. MD5(md5\_input\_val, md5\_input\_length, MD5\_hash\_result\_buffer);
7. }

**Insert函数**

* 先调用hash函数
* 将md5\_hash\_result\_buffer中的内容通过强制类型转换，转换成2字节的无符号整数。
  + - 由于md5\_hash\_result\_buffer是16字节的，所以最多可以有8个独立的2字节无符号整数，故哈希函数最多可以选取8个。
    - 并且如此以来，实际上哈希函数的计算只有1次（调用md5），简化了多个哈希函数的计算量。
* 将k个哈希值对印位置设为1.
  + - 由于哈希值的范围是2字节也就是2^(8\*2)=2^16次方，若哈希值的大小大于了哈希表的大小会错误，需要将哈希值对哈希表大小取模。

1. **void** insert(**const** std::string& object)
2. {
3. hash(object);
4. **const** u\_int16\_t\* **const** object\_hashes = **reinterpret\_cast**<**const** u\_int16\_t \* **const**>(MD5\_hash\_result\_buffer);
6. **for** (**size\_t** i = 0; i < hash\_function\_count; i++)
7. {
8. **const** u\_int16\_t index\_to\_set = object\_hashes[i] % bloomfilter\_store\_size;
9. bloomfilter\_store\_[index\_to\_set] = **true**;
10. }
11. ++object\_count\_;
12. }

**Contains函数**

* 查询该元素时，与insert同理。
* 如果k个位置均为1，则说明元素存在。如果有一个为0就说明不存在。

1. **bool** contains(**const** std::string& object) **const**
2. {
3. hash(object);
4. **const** u\_int16\_t\* **const** object\_hashes = **reinterpret\_cast**<**const** u\_int16\_t \* **const**>(MD5\_hash\_result\_buffer);
6. **for** (**size\_t** i = 0; i < hash\_function\_count; i++)
7. {
8. **const** u\_int16\_t index\_to\_get = object\_hashes[i] % bloomfilter\_store\_size;
9. **if** (!bloomfilter\_store\_[index\_to\_get]) **return** **false**;
10. }
11. **return** **true**;
12. }

## 核心函数的名称和功能

Main函数一共完成了2个任务

* 验证bloomfilter的正确性
  + Main中未被注释的部分
* 不断调整m的大小，探索错判和消耗空间之间的关系
  + Main中被注释的部分
    - 探索了m的大小从20000增长到67000过程中错判的数量（使用的是f10000.txt的数据集合）

## 输入输出的格式等

在程序的第12行和第13行更改 file\_name, query\_name， 使用fstream的open方法打开文件，因此绝对路径或者是相对路径均可以。

1. string file\_name[2] = {"../Week11\_Data/f100.txt","../Week11\_Data/f10000.txt"};
2. string query\_name[2] = {"../Week11\_Data/q100.txt","../Week11\_Data/q10000.txt"};

read\_file函数中，设置index为1，打开的是f10000.txt和q10000.txt，同时还将a10000.txt读入作为答案的对照。所以还需在程序的第78行**更改a10000.txt的路径**。

1. fin.open("../Week11\_Data/a10000.txt");

默认将结果输出到同路径下的ans.txt文件中。

# 四、实验结果说明：

## 实验结果截图

编译命令中的-I和-L部分是我下载的第三方的库的位置（参考阅读资料的编译命令的写法）

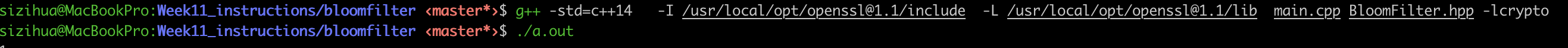


Figure ：编译并运行程序

Mac系统下使用brew安装该库。命令为 brew install openssl

安装成功后，用brew link查看链接对应的位置

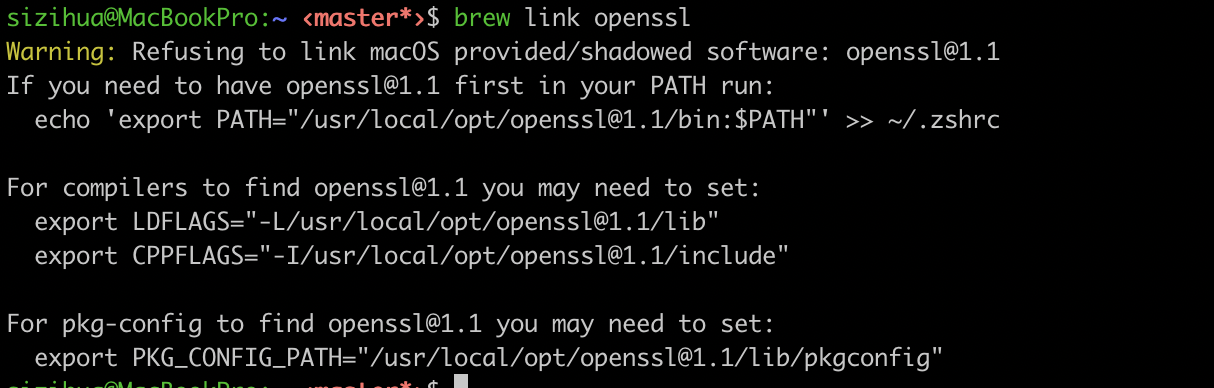


Figure ：编译链接位置

下面的两个图分别是运行数据集1和数据集2的结果。

使用4个独立哈希函数，使用2^16大小的哈希表进行的测试。数据集1得到的结果全部正确，而数据集2中有个别false negative。

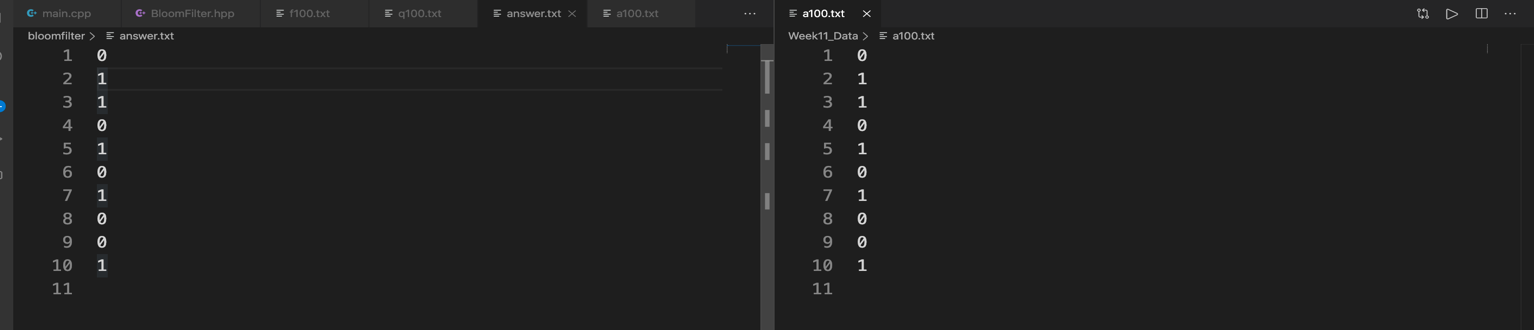


Figure ：数据集1的运行结果

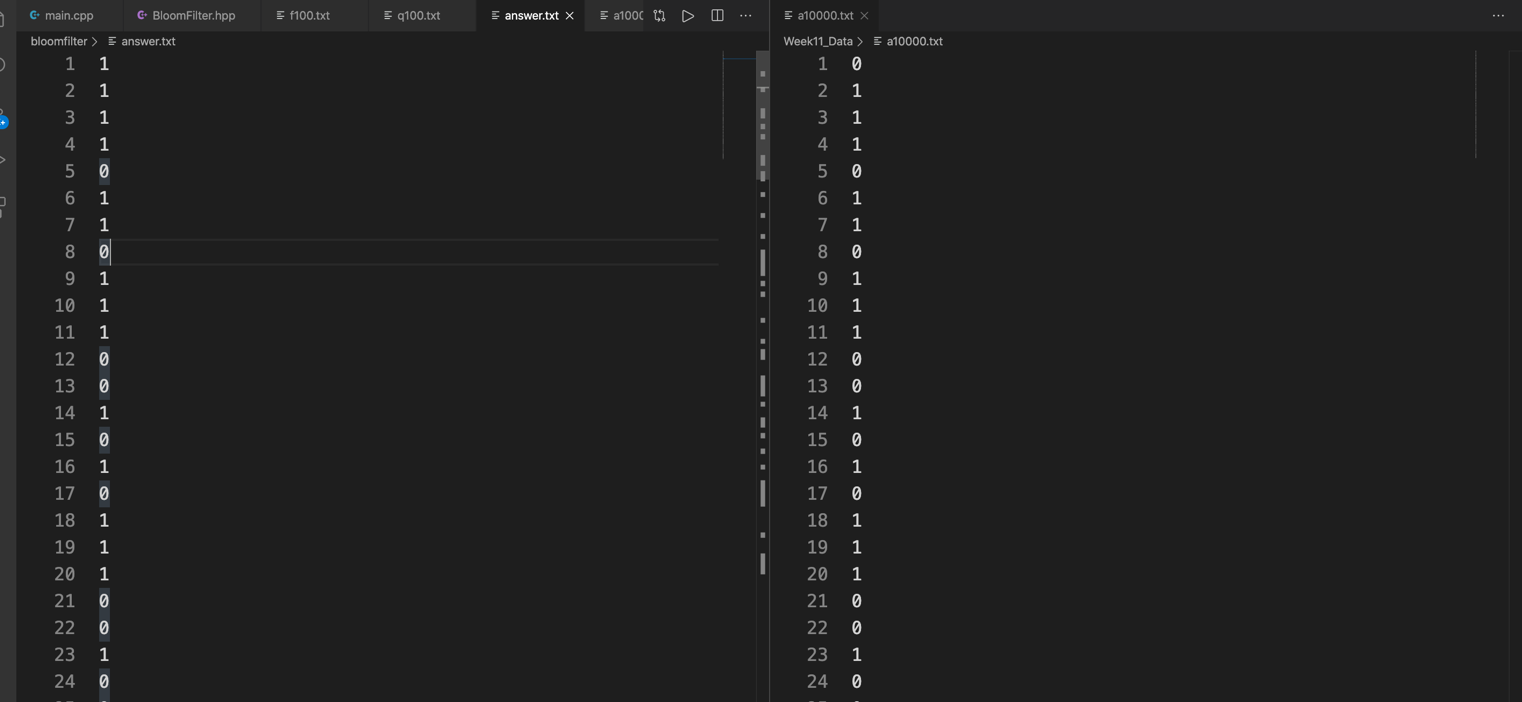


Figure ：数据集2的运行结果

## False negative和空间大小关系探究

使用数据集2，即元素个数n=10000的数据集。

探究m大小从20000到67000的过程中，false postive的变化情况。

在此过程中始终令k =

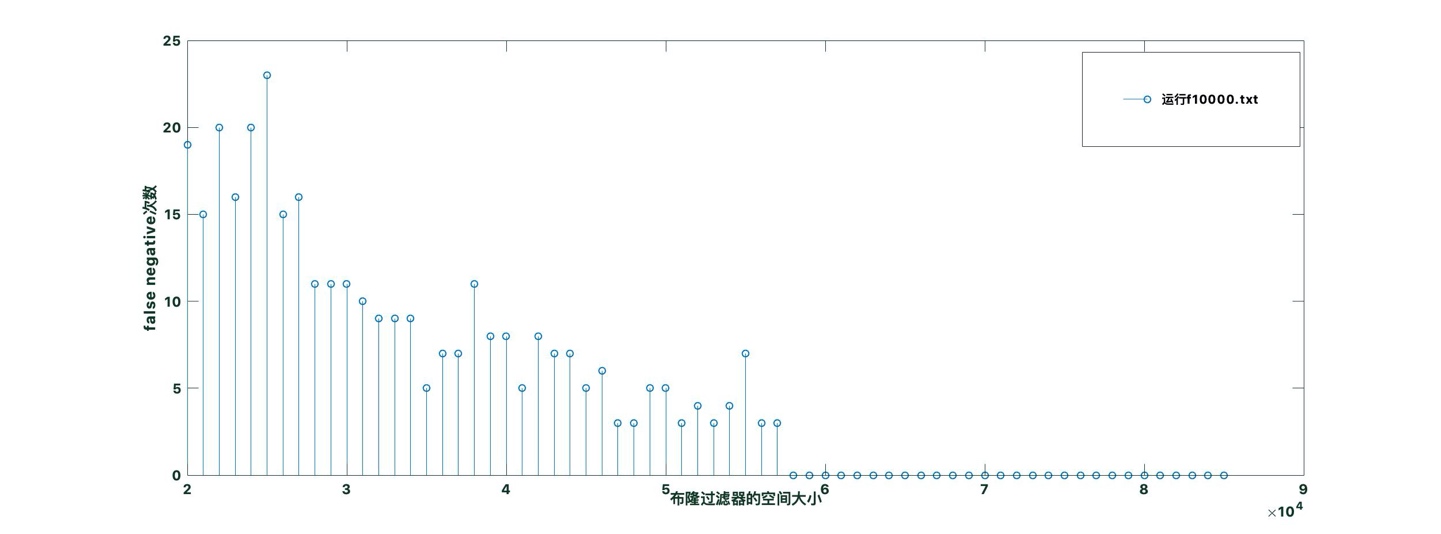


Figure :探究结果

探究结果如Figure 5所示。可以看出随着m的增大false postive的数量在减小。

**理论分析false postive和m的关系**

令k =

计算可得Pr(fasle positive)

实验中探究部分n固定为10000，m不断增大，所以Pr(false positive)不断减小，这和实验结果相符。