

“程序设计与算法训练”

课程设计报告

**设计题目**  布隆过滤器的实现和应用

**姓 名**  温嘉昊

**学 号**  2018218794

**专 业**  计算机科学与技术

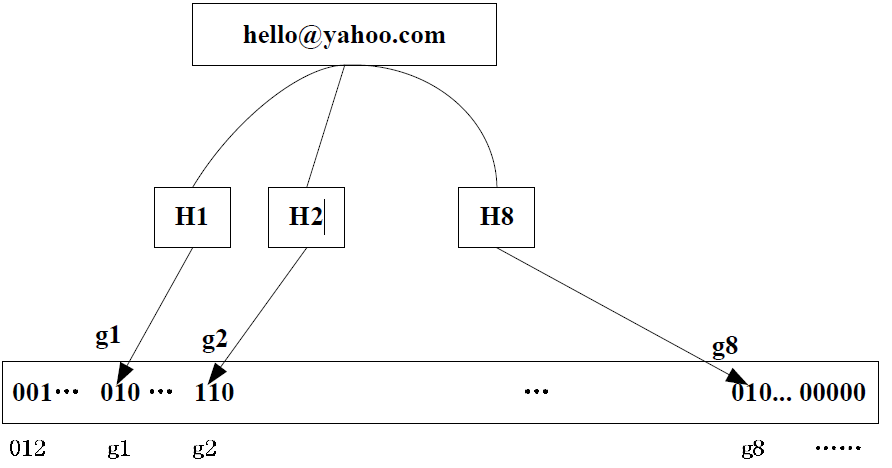
**班 级**  计算机18-3班

**完成日期**  2020年2月14日

**（一） 需求和规格说明**

（1）问题描述

布隆过滤器是由巴顿.布隆于一九七零年提出的。它实际上是一个很长的二进制向量和一系列随机映射函数。假定需要存储一亿个电子邮件地址，首先建立一个十六亿二进制（比特），即两亿字节的向量，然后将这十六亿个二进制全部设置为零。对于每一个电子邮件地址X，可以用八个不同的散列函数（H1,H2,...,H8）产生八个从1到十六亿之间中的八个自然数g1,g2,...,g8。然后将这八个自然数对应的八个位置的二进制全部设置为一。同样的方法对这一亿个email地址都进行处理后，一个针对这些email地址的布隆过滤器就建成了。如图所示：



现在看看如何用布隆过滤器来实现一个电子邮件地址过滤器，首先将那些放在黑名单上的电子邮件地址放在布隆过滤器中。当检测一个可疑的电子邮件地址Y是否在黑名单中，仍用相同的八个随机数产生器（H1,H2,...,H8）对这个邮件地址产生八个自然数s1,s2,...,s8，这八个自然数对应的八个二进制位分别是t1,t2,...,t8。如果Y在黑名单中，显然，t1,t2,..,t8对应的八个二进制一定是一。这样在遇到任何在黑名单中的电子邮件地址，都能准确地发现。

布隆过滤器决不会漏掉任何一个在黑名单中的可疑地址。但是，它有一条不足之处。也就是它有极小的可能将一个不在黑名单中的电子邮件地址判定为在黑名单中，因为有可能某个好的邮件地址正巧对应个八个都被设置成一的二进制位。但这种可能性很小，此处将它称为误识概率。在上面的例子中，误识概率在万分之一以下。

因此布隆过滤器的好处在于快速，省空间。但是有一定的误识别率。常见的补救办法是在建立一个小的白名单，存储那些可能别误判的邮件地址。

（2）课程设计目的

学习BloomFilter结构，能应用该结构解决一些实际问题。

（3）基本要求

①定义BloomFilter结构的ADT，该ADT应支持在BloomFilter中加入一个新的数据，查询数据是否在此过滤器中，并完成该结构的设计和实现。

②应用BloomFilter结构拼写检查，许多人都对Word的拼写检查功能非常了解，当用户拼错一个单词的时候，Word会自动将这个单词用红线标注出来。Word的具体工作原理不得而知，但另一个拼写检查器UNIXspell-checkers这个软件中就用到了BloomFilter。UNIXspell-checkers将所有的字典单词存成BloomFilter数据结构，而后直接在BloomFilter上进行查询。本课程设计要求针对C语言设计和实现上述拼写检查器，即当写了一个正确的关键词，如int时，给该词标上颜色，如蓝色。

③针对上述C语言关键词拼写检查器进行分析，如错误分析，设计散列函数个数分析，运行时间复杂性、空间复杂性的分析。

④上述C语言关键词拼写检查器最好是在VC++或Java等可视化开发环境下实现。

⑤上述C语言关键词拼写检查器最好能支持所有的C++关键词。

**（二） 设计**

根据上述需求，要想实现拼写检查，即需要建立一个布隆过滤器，当对文本逐词进行遍历时，发现存在于字典中的关键字，即需要设计一个类Bloomclass，作为BloomFilter结构的ADT,主要功能用于存入字典关键词和验证文本词句是否在过滤器内。当需要存入字符串时，利用addString()函数调用8个哈希散列函数对字符串进行8次特征位计算，并将BloomFilter结构的对应二进制位置1，同样可利用checkString()函数检查某字符串经计算后是否对应二进制位皆为1。由于布隆过滤器具有一定误识别率，因此额外还设置了一个白名单排除函数excludestring()，当字符串在白名单中时，及时二进制位置皆为1也不视为关键字。同时，由于布隆过滤器本身并不储存元素本身且删除困难，为了方便查看已经储存的字符串，定义了importdict()函数将文件中的字符串读取到布隆过滤器中，exportdict()函数来将布隆过滤器所储存的字符串储存到文件中。

同时，为了将内容模块化，将所有涉及程序和文件交互、与用户交互无关的功能函数如openfile()打开文件，formatfile()将文本框中关键词彩色标出，Opendictfile()选取包含关键词的字典文件等函数定义在MenuCommand类中，这样可以保证源码功能的专一性，也方便了程序从控制台界面到图形界面的转换。另外，二进制储存文件设置为以下格式：



同时，为了方便日后了解二进制储存文件中包含的关键字，将所有包含的关键字储存在关键词文件中，该类文件格式如下：

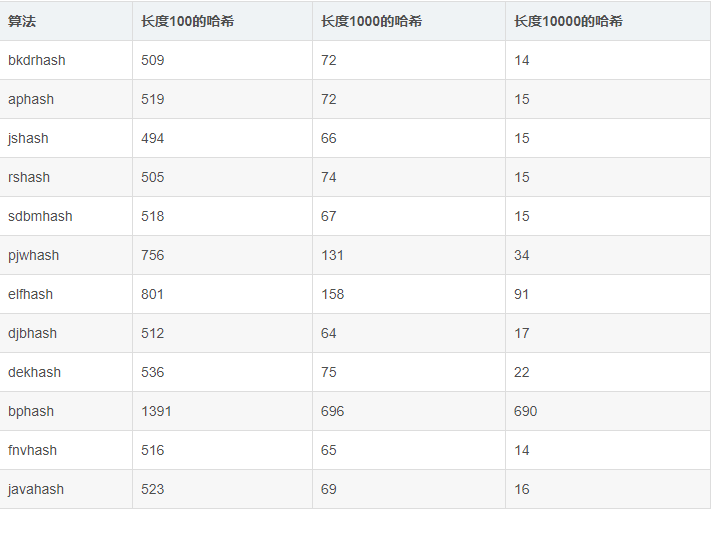


另外，为了将结果进行可视化，需要设置图形界面，在此，由于C++支持使用Win32 API函数库，因此可以使用库中的Windows Controls系列函数制作简单的图形界面，同时为了在同一个文本框中同时显示多种样式的字体，需要使用支持XML格式的富文本框。为了方便MenuCommand类中的函数将读取到的文本显示到图形界面中的富文本框中，定义了textcontent类作为读取和设置富文本框内容的类，主要用于界面和功能函数的交互。主要包含按空格取出字符串的函数findwordbreak()，设置词汇字体颜色的函数turnblue()将部分字体变为蓝色和turnnormal()将部分字体变为黑色，

在哈希散列函数的设计过程中，散列函数的选取十分重要，由于要实现C++的49个关键字，根据公式

*(m为二进制位数量，p为误判率，n为将要加入集合的元素个数)*

为了使布隆过滤器发挥作用，误判率p需要小于1%，n取49，得到m的值为325.6，即m至少取326时才能发挥布隆过滤器的功能并且保证低误差率，取m为512，我使用的比特位在100-10000位之间经过对比，如图中所示：

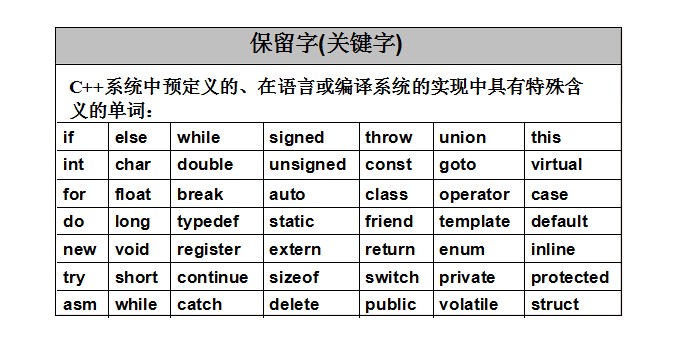


*各种散列算法在哈希表长度为100,1000和10000时对45402个单词的最大冲突数*

我最终选取了在长度为1000左右时冲突数较小的bkdrhash算法、aphash算法、jshash算法、rshash算法、sdbmhash算法、djbhash算法、fnvhash算法和javahash算法。

1. 时间复杂度：
2. 初始化： 该布隆过滤器对给定二进制向量长度大小的内存进行初始化都需要从头到尾置零，而我采用short类型来储存二进制位，故该布隆过滤器初始化和释放内存的时间为o(m/16+1) （m为二进制向量的维数），时间复杂度为o(m)。
3. 检测或添加一个关键词： 由于该布隆过滤器对每个字符串都要从头到尾进行遍历，因此该布隆过滤器添加一个字符串或对一个待检测字符串进行检验的时间为 o(8n) (n为待检测字符串的长度，其中的8为散列函数的个数)，时间复杂度为o(n)。
4. 空间复杂度：
5. 初始化： 由于该布隆过滤器初始化需要占用(m/16+1) \* sizeof(short) + 3 \* sizeof(int) = [m/8] + 14字节大小的空间, 其中m为二进制向量的维数。空间复杂度为o(m)。
6. 检测或添加一个关键词： 进行检测或添加一个关键词需要利用13 \* sizeof(int) + 16 \* sizeof(short) = 104字节大小的空间。空间复杂度为o(1)。

C++的关键字共49个，如图：



为了对该布隆过滤器进行散列函数的分析，需要先假定所有的散列函数形式不同，但对于同样个数的字符串产生冲突的概率相同，且同一字符串对不同散列函数的性能影响相同。在上述情况下，只需要保证要检测的字符串相同，关键词相同，通过改变不同情况下的个数，即可得出相关结论。为了先确定散列函数个数的大致范围,根据公式：

*（k为散列函数个数，m为布隆过滤器的大小，n为将要加入集合的的元素个数）*

将m=512，n=49代入公式可得散列函数个数大概为7个。

为了对该布隆过滤器进行错误分析，可以给定一串字符串，确定其误差率，根据真实误差公式：

*( 为真实误差率，m为二进制位数量，k为散列函数个数，n为将要加入集合的元素个数)*

将m=512，n=49，k=7代入公式可得其真实误差率为0.6%。

**系统类图**

bloomclass

bloom bloomfilter

int hashfunction\_num

bool hash5(char\*, int)

bool hash6(char\*, int)

bool hash7(char\*, int)

bool hash8(char\*, int)

bool hash9(char\*, int)

bool hash10(char\*, int)

bool hash11(char\*, int)

bool hash12(char\*, int)

int GetBloomlen ()

int SetBloomlen (int)

bool checkstring (char\*)

bool AddString(char\*)

int Initspellchecker(int)

int exportdict(TCHAR\*)

int importdict(TCHAR\*)

int releasedict()

bool checkbit(int position)

bool turnbit(int position)

void sethashfunction\_num(int)

short gethashfunction\_num()

MenuCommand

TCHAR GLOBALFILENAME[PATHMAXLEN]

TCHAR GLOBALRUNNINGPATH[PATHMAXLEN]

TCHAR GLOBALDICTFILENAME[MAX\_PATH]

TCHAR publicbuffer[MAXLEN]

int OpenFilef()

int SaveFilef ()

int FormatFilef ()

int OpenDictFilef()

int savedictfile()

int NewFilef()

void sethashfunction\_num(int)

textcontent

CHARFORMAT2 normalstyle

CHARFORMAT2 specialstyle

void settext(TCHAR\* income)

void cleartext()

void gettext(TCHAR\* tcharbuffer,int re\_text\_length)

void getwordrange(textrange& textrangevar)

void turnblue(\_charrange rangevar)

void turnnormal(\_charrange rangevar)

void adddictword(TCHAR\* tcharbuffer)

void getdictword(int index,TCHAR\* tbuffer)

int getlength()

int getlengthEX()

int findnextbreak(int startposition)

int getdictwordnumber()

int getdictwordlen(int index)

FileDialog

TCHAR\* Getfilepath(bool isopen)

bool OpenWindowsDlg (bool IsOpen, bool IsPickFolder, TCHAR\* pFilePath)

**属性和方法定义**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **类名** | **成员类别** | **类型** | **成员名** | **描述** |
| bloomclass | 属性 | bloom | bloomfilter | 布隆过滤器数据结构，主要包括一个unsigned int\*指向二进制向量 |
| int | hashfunction\_num | 需要使用的哈希函数个数 |
| 公有方法 | int | GetBloomlen () | 获取布隆过滤器的大小，即二进制位数目 |
| int | SetBloomlen (int) | 设置布隆过滤器的大小，即二进制位数目 |
| bool | checkstring (char\*) | 添加目标字符串到布隆过滤器 |
| bool | AddString(char\*) | 检查目标字符串是否在布隆过滤器 |
| int | Initspellchecker(int) | 初始化布隆过滤器 |
| int | exportdict(TCHAR\*) | 将布隆过滤器结构体的二进制位数据导出到目标文件 |
| int | importdict(TCHAR\*) | 向布隆过滤器导入来自指定路径文件内的二进制位数据 |
| int | releasedict() | 释放布隆过滤器所占用的内存 |
| 私有方法 | bool | hash5(char\*, int) | BKDRhash散列函数，如果int参数为1则将二进制向量对应标志位置1，否则跳过。之后查询该位置是否为1 |
| bool | hash6(char\*, int) | APhash散列函数，如果int参数为1则将二进制向量对应标志位置1，否则跳过。之后查询该位置是否为1 |
| bool | hash7(char\*, int) | DJBhash散列函数，如果int参数为1则将二进制向量对应标志位置1，否则跳过。之后查询该位置是否为1 |
| bool | hash8(char\*, int) | JShash散列函数，如果int参数为1则将二进制向量对应标志位置1返回值为该位置是否为1 |
| bool | hash9(char\*, int) | RShash散列函数如果int参数为1则将二进制向量对应标志位置1，否则跳过。之后查询该位置是否为1 |
| bool | hash10(char\*, int) | SDBMhash散列函数，如果int参数为1则将二进制向量对应标志位置1，否则跳过。之后查询该位置是否为1 |
| bool | hash11(char\*, int) | FNVhash散列函数，如果int参数为1则将二进制向量对应标志位置1，否则跳过。之后查询该位置是否为1 |
| bool | hash12(char\*, int) | Java字符串散列函数，如果int参数为1则将二进制向量对应标志位置1返回值为该位置是否为1 |
| bool | checkbit(int position) | 将输入的位置转换为二进制位的位置并检验该位置比特位是否为1 |
| bool | turnbit(int position) | 将输入的位置转换为二进制位的位置并将该位置比特位置1 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **类名** | **成员类别** | **类型** | **成员名** | **描述** |
| FileDialog | 方法 | TCHAR\* | Getfilepath(bool isopen) | 建立一个文件浏览框用于返回文件路径，取消返回空。其中isopen取FILE\_DIALOG\_CMD\_OPEN时为打开文件，为FILE\_DIALOG\_CMD\_CREATE时为保存文件。 |
| bool | OpenWindowsDlg (bool IsOpen, bool IsPickFolder, TCHAR\* pFilePath) | 高级的浏览框，isopen取FILE\_DIALOG\_CMD\_OPEN时为打开文件，为FILE\_DIALOG\_CMD\_CREATE时为保存文件。IsPickFolder取TRUE时为选取文件夹，FALSE时为选取文件，pFilePath用于保存选中文件的路径用于返回。 |
| MenuCommand | 属性 | TCHAR | GLOBALFILENAME[PATHMAXLEN] | 文件路径，用于储存目前读取的文件路径，退出为空。 |
| TCHAR | GLOBALRUNNINGPATH[PATHMAXLEN] | 运行路径，用于储存编译和运行代码的文件夹路径 |
| TCHAR | GLOBALDICTFILENAME[MAX\_PATH] | 字典文件路径，用于储存目前选定的字典文件路径。 |
| TCHAR | publicbuffer[MAXLEN] | TCHAR类型的临时字符串储存区，可用于同一类中不同函数间相互传递信息。 |
| 方法 | int | OpenFilef() | 打开文件，调用此函数来显示一个文件内容到富文本框。 |
| int | SaveFilef () | 保存文件，调用此函数来将富文本框显示的内容保存到一个文件中 |
| int | FormatFilef () | 格式化文件，调用此函数以将字典中的关键词以彩色的文字显示出来。 |
| int | OpenDictFilef() | 编辑字典文件 |
| void | savedictfile() | 保存词典文件 |
| int | NewFilef() | 新建文件，清除屏幕上的文本内容 |
| int | CloseFilef() | 关闭文件，关闭文件并清除屏幕上的文字 |
| int | CompilerFilef() | 编译代码，将文件视为源代码进行编译 |
| int | GeneraterFilef() | 生成代码，将文件视为源代码进行生成 |
| int | RunningCodef() | 运行代码，将文件视为源代码运行 |
| int | RefreshRunningCodef() | 重新编译运行，将文件视为源代码编译并运行。 |
| int | openexistdictfile() | 编辑已经存在的字典文件 |
| int | opennewdictfile() | 编辑新的字典文件 |
| textcontent | 属性 | CHARFORMAT2 | normalstyle | 黑色的字体样式 |
| CHARFORMAT2 | specialstyle | 蓝色加粗的字体样式 |
| 方法 | void | settext(TCHAR\* income) | 添加文本到富文本框中 |
| void | cleartext() | 清除富文本框中的文本 |
| void | gettext(TCHAR\* tcharbuffer,int re\_text\_length) | 获取指定re\_text\_length长度的文本到tcharbuffer中 |
| void | getwordrange(textrange& textrangevar) | 获取指定范围的文本 |
| void | turnblue(\_charrange rangevar) | 将指定范围的文本变为蓝色 |
| void | turnnormal(\_charrange rangevar) | 将指定范围的文本变为黑色 |
| void | adddictword(TCHAR\* tcharbuffer) | 添加tcharbuffer中的关键词到列表 |
| void | getdictword(int index,TCHAR\* tbuffer) | 获取列表中下标为index的关键词文本到tbuffer中 |
| int | getlength() | 利用系统函数获取控件文本长度 |
| int | getlengthEX() | 将文本视为CP\_ACP编码的文本并获取长度 |
| int | findnextbreak(int startposition) | 寻找startposition之后第一个文本分隔符的位置 |
| int | getdictwordnumber() | 获取列表中关键词的数量 |
| int | getdictwordlen(int index) | 获取列表中下标为index的关键词的长度 |

**（三） 用户手册**

程序启动后，可以看到菜单栏分为“文件”“编辑”“项目”“帮助”四个选项，“文件”选项中包含新建文件、打开文件、保存文件、关闭文件、退出程序功能，利用这些功能可以导入和导出富文本框中的内容。

“编辑”选项中包含格式化富文本框内容（利用布隆过滤器标记其中的内容）、导入字典文件、编辑字典文件。

“项目”选项中包含编译运行、编译、运行功能，可将文本文件作为代码编译运行。“帮助”中包含关于，可以查看程序相关信息。

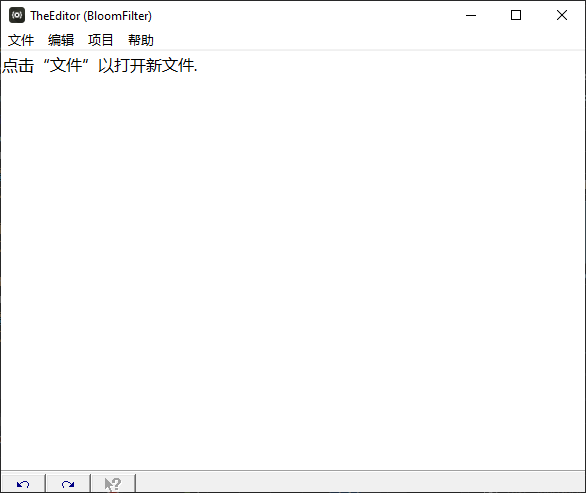
1. 初次运行可以用“文件->新建”打开文件，编写程序。
2. 可以用“文件->保存”保存文件。
3. 利用“编辑->格式化”，并找到“stdc++.bin”字典文件，导入后，再次点击“编辑->格式化”将文字中的c++关键字高亮显示出来。
4. 利用“项目->编译运行”可以将代码编译并运行。

**（四） 调试及测试**

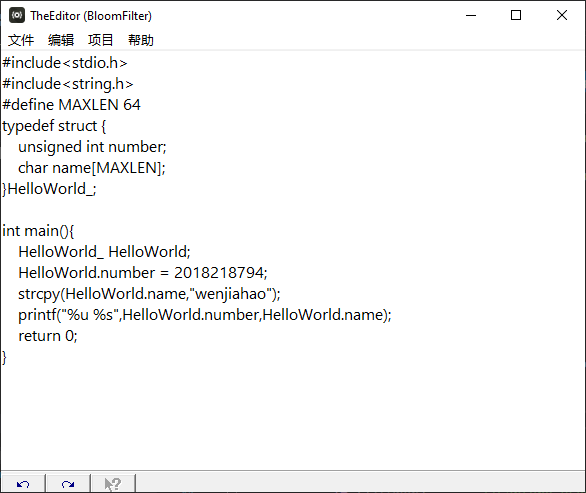
由于程序每次运行在读取完字典文件后才可以进行格式化，因此格式化前需先检查是否导入字典文件，如没有则先导入。

在读取文件后，需要进行多次操作，因此可以将文件路径等信息保存为类的静态变量，方便不同函数的调用。

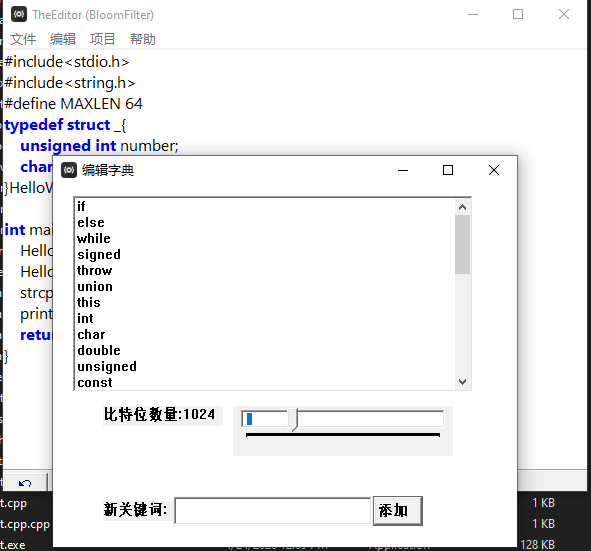
**（五） 运行实例：**



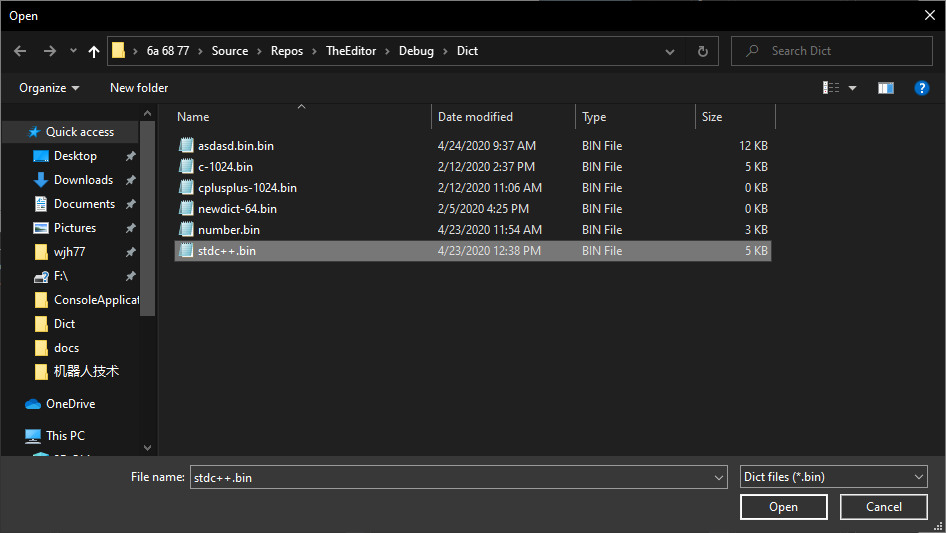
*图-1 程序主界面*



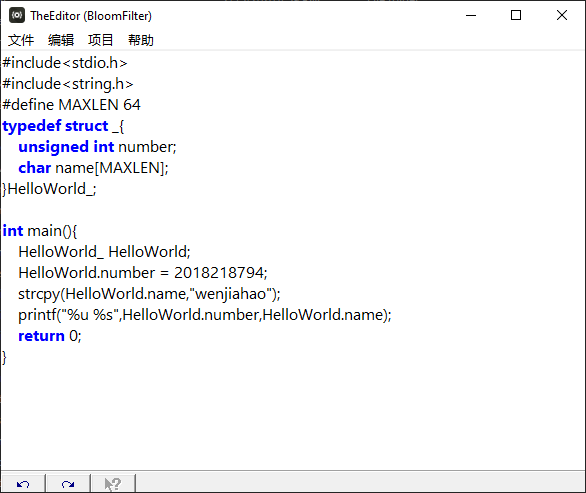
*图-2 利用“文件->新建”新建文件并编写代码*



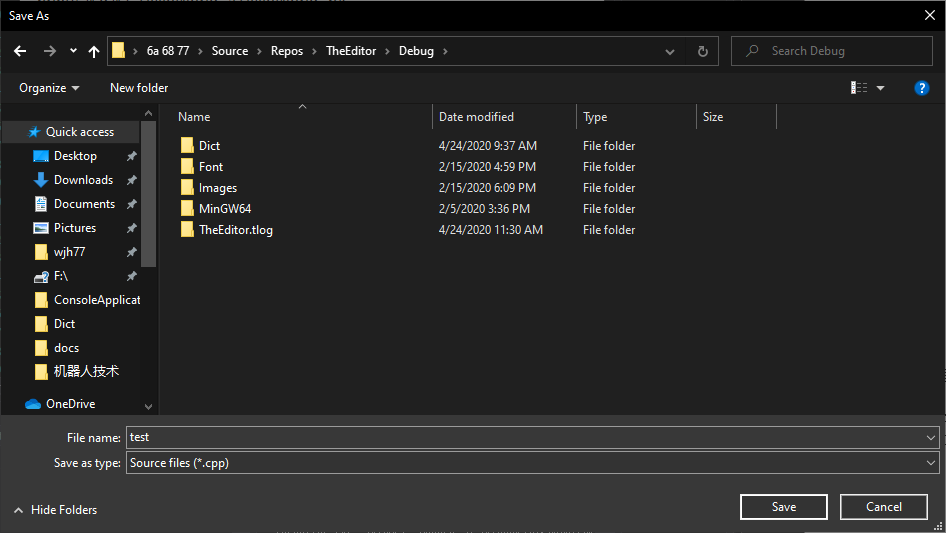
*图-3 利用“编辑->编辑字典文件”查看字典文件已包含的关键字。*



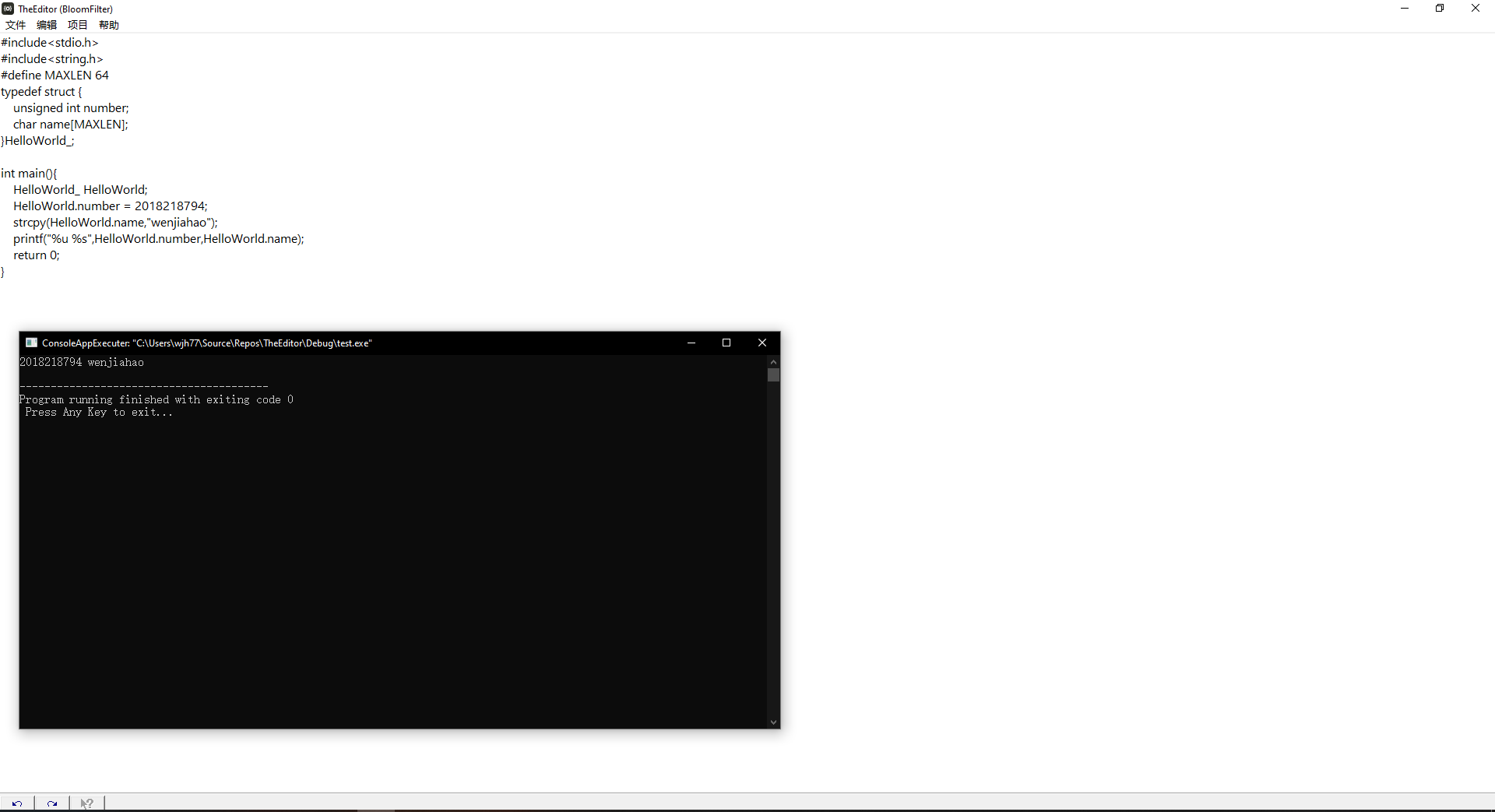
*图-4 选取字典文件*



*图-5 利用“编辑->格式化”高亮显示代码。*



*图-6 利用“文件->保存”保存文件到本地。*



*图-7 利用“项目->编译运行”编译运行已编译完成的代码。*

**（六）进一步改进**

（1）目前程序中，白名单无法手动添加，只能通过源码中已存在的白名单进行鉴别，这无疑降低了布隆过滤器的通用性，可以考虑在编辑字典文件时添加白名单列表和编辑窗口。

（2）目前程序中，只做到了布隆过滤器内关键词的添加和查看，无法进行删除操作，可考虑在编辑字典文件时添加删除操作使得程序容错率更高。

（3）目前程序显示文本主要依靠richtextbox控件实现，然而richtextbox具有诸多缺点如：内容大小不能超过64K，缺少固定滚动条的函数，在处理大量的代码时会造成页面滚动，可考虑使用其他开源控件如RRichTextBox或freeTextBox

**（七）心得体会**

通过本次课程设计，我了解了布隆过滤器的工作机理及其简单应用。也认识到数据结构这门课程是计算机学科的一门必不可少的课程，通过课上所学，可以实现许多算法的很多问题。在本次课程实验中，我利用了顺序表来储存二进制向量，以及使用哈希散列函数产生关键词的特征位，利用课程思想，可以灵活地解决很多问题。

在课程设计过程中，我同样也遇到了以前从未遇到过的诸多问题，这也反映了要将自己的所得所学应用到实际生活中，仍然需要一定的新方法和新思想。

我首先遇到的问题，是在将布隆过滤器的数据进行特征位运算的问题，由于布隆过滤器是根据特征位进行比较，而C/C++语言中最小数据单位char具有8位长度，要想直接进行比较是不可能的，这时就需要利用位运算来解决问题。在实际使用位运算的过程中，将一个数据进行与1进行或运算设置成1后，在进行判断该数据的值时，我想当然的继续利用与1进行或运算，然而这样做的结果却并不正确，经过数次尝试后，我终于总结得出，或运算进行赋值运算后，想要比较该数据的值必须通过与运算。这也打消了我希望将检验位是否置1的函数和将某位置1的函数合二为一以减少代码量的想法，在某些情况下，代码的确不能节省。

其次，为了将二进制位储存到文件中以便下次直接导入，我实现了向文件输出二进制向量的函数，然而当我想要读取时，却发生了问题，如何保证文件是完整的并且正确的？为了防止某次操作使用了错误的文件造成程序崩溃或文件损坏，保证读取到的词典文件和布隆过滤器的二进制位文件的完整性和正确性是十分重要的。为此我后来采用将文件的前几个字节设置为“bfd”及“bfdc”的方法来防止这种事件的发生。

在设置布隆过滤器时，如何灵活的设置布隆过滤器的关键词？由于将关键词添加到布隆过滤器的过程是单向不可逆的，因此要想将布隆过滤器的优点发挥出来，并且提供删除关键词，观察存在的关键词的功能，我设法设置了另外一个文件用于储存已经添加的关键词。并利用listbox显示这些关键词。这个文件和储存二进制向量的文件文件名相同，拓展名为“bin\_config”和“bin”。当在编辑关键词后，程序除了将关键词保存到“bin\_config”文件外，还会对所有关键词重新产生二进制向量并保存到“bin”文件中，这样可以极大增加布隆过滤器的灵活性。

在将文本框中的单词和布隆过滤器进行比较时，如何将richtextbox控件中的文字提取成单词并进行与过滤器的比对？如果将文字全部提取再进行判断会十分繁琐，将richtextbox中的文本全部提取出来的方法有发送消息EM\_STREAMOUT或getwindowtext()但二者都需要一定的空间储存提取出的字符串。经过查阅资料，我发现了系统中针对文本框提供了多种函数，其中我采取了利用richtext的取词函数，通过发送消息EM\_FINDWORDBREAK，从而不断获取词的起始位置，又由于单词间彼此连续，故可间接得到每个单词的始末位置，从而得到单词进行比对。

事实上，最令我困惑的问题，在于feof()函数的使用。在我设计将关键词保存到文件的函数和从文件中读取关键词的函数时，我利用系统函数从列表框中获取到TCHAR[]类型的关键词，再用malloc函数分配一个与关键词大小相同的char[]字符数组，通过wcstombs\_s()函数将TCHAR字符串复制成CHAR字符串后，再用fwrite()函数将其写入到文件中，在这个过程中，为了使得读取关键词时更简单，我设置了fwrite()写入时对于每个关键词都写入MAXLEN长度的字符，这样一来，在读取关键词时，只需要每次fread()函数读取MAXLEN长度的数据，即可获得一个关键词。然而经过实践，我发现这样写入虽然没有问题，然而读取时却会出现关键词读取不全，或者最后一个关键词多次重复的问题。由于我使用feof()函数作为判断文件是否读取结束，因此我很容易就认为是feof()函数存在问题，经过查阅资料，我终于认识到，feof()函数只能判断文档是否达到结尾，他所判断文档是否打到结尾的标志，是依靠特殊的二进制标志-1，而在所有的关键词皆为可见字符的前提下，出现-1这种字符的原因，只可能出现在写入文件时写入了EOF标志的数据，即写入了未经初始化的数据。在写入关键词到文件的过程中。malloc分配的内存，往往小于fwrite()写入的内存，这就意味着文件的读取可能会发生错误。通过改变TCHAR[]和CHAR[]为固定大小MAXLEN，最终消除了这个错误。

在实验后的调试过程中，我也开始逐渐意识到即使计算机发展迅速，C++作为面向对象的语言对硬件的要求逐渐降低，然而内存的分配和管理仍然不能忽视，由于变量大小不够会导致函数调用出现异常，导致程序崩溃。

经过本次课程设计，使我从C语言的面向过程思维逐渐向面向对象转变，我开始认识到要解决问题需要的不仅仅是考虑整个算法。更重要的是考虑整个程序流程，先搭建好骨架，在对每个功能函数逐个进行设计，这样做不仅仅可以使自己以后再做该项目时不用再从头看起，也可以使自己能够再想到一个问题更简便的解法时，能够准确增减代码，做到精准无误。

**（八）对课程设计的建议**

课程设计的题目都很新颖，贴合实际，能够使我在编程中学习到一些新知识，为我以后的成长提供一定的技术支撑。在编程中，尽管课程不要求使用图形化编程实现，但我还是希望能在课程设计之前开设一些关于图形化编程方面一些课程，这样可以使我更轻松地掌握图形编程的艺术和方法。

**（九）附录⎯⎯源程序**

###### 1. 核心代码

#### BloomClass.h

#pragma once

#ifndef BLOOMCLASS\_H

#define BLOOMCLASS\_H

#include"TheEditorcommon.h"

typedef struct bloomstruct {//布隆过滤器数据结构

unsigned short\* dicthash = NULL;//比特位开头指针

int BLOOMFILTERBITCOUNT = 0;//比特位数量，即

int allocatesize = 0;//分配的short类型内存数量

}bloom;

class Bloomclass {

public:

Bloomclass(int);//构造函数

~Bloomclass();//析构函数

int GetBloomlen();//获取布隆过滤器比特位数量

int SetBloomlen(int); //设置布隆过滤器比特位数量

bool checkstring(char\*);//检查字符串的特征位是否在布隆过滤器中全部为1

bool AddString(char\*); //添加字符串的特征位到布隆过滤器

int initspellchecker(int);//初始化布隆过滤器

void echodicthash();//向字符缓冲区输出布隆过滤器中每八个比特位对应的十六进制数值

int exportdict(TCHAR\*);//导出所有比特位和相关信息到目标文件

int importdict(TCHAR\*);//解析导入目标文件内容并填充所有比特位

int releasedict();//重置布隆过滤器

void sethashfunction\_num(short num) {//设置使用的散列函数个数

hashfunction\_num = num;

}

short gethashfunction\_num() {//获取使用的散列函数个数

return hashfunction\_num;

}

private:

bloom bloomfilter;

char tmpbuffer[MAXLEN]= "";//临时字符

char ltmpbuffer[MAXLEN] = "";//临时字符

char lltmpbuffer[MAXLEN] = "";//临时字符

char llltmpbuffer[MAXLEN] = "";//临时字符

char lllltmpbuffer[MAXLEN] = "";//临时字符

char llllltmpbuffer[MAXLEN] = "";//临时字符

static short hashfunction\_num;//散列函数数量

bool excludeString(char\* buffer);//检查字符串是否在白名单中

bool checkbit(int position);//检查某比特位的值

bool turnbit(int position);//设置某比特位的值

bool checkbit\_v1(int position);//检查某比特位的值

bool turnbit\_v1(int position);//设置某比特位的值

bool hash1(char\*, int);//散列函数

bool hash2(char\*, int);//散列函数

bool hash3(char\*, int);//散列函数

bool hash4(char\*, int);//散列函数

bool hash5(char\*, int);// BKDR 散列函数

bool hash6(char\*, int);// AP 散列函数

bool hash7(char\*, int);// DJB 散列函数

bool hash8(char\*, int);// JS 散列函数

bool hash9(char\*, int);// RS 散列函数

bool hash10(char\*, int);// SDBM 散列函数

bool hash11(char\*, int);// FNV 散列函数

bool hash12(char\*, int);// JAVA 散列函数

};

#endif

#### Bloomclass.cpp

#include"BloomClass.h"

short Bloomclass::hashfunction\_num = 8;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称: Bloomclass::hash1(char\* buffer, int command = 0)

函数功能: 将字符串按照ascii码变为数字，并得到其积，最后对比特长度取余，如果int参数为1则将bloomfilter对应标志位置1，否则跳过。之后查询该位置是否为1.

传入参数: char\*目标字符串，int 0：查询，1：更改

返回值: 如果是1返回true，否则为false。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool Bloomclass::hash1(char\* buffer, int command = 0) {//Deprecated

unsigned int tmp = 1;

while (\*buffer != '\0')

{

tmp \*= (int)(\*buffer);

tmp %= bloomfilter.BLOOMFILTERBITCOUNT;

buffer++;

}

if (command == 1)

return turnbit(tmp % bloomfilter.BLOOMFILTERBITCOUNT);

return checkbit(tmp % bloomfilter.BLOOMFILTERBITCOUNT);

//\*(bloomfilter.dicthash + tmp % bloomfilter.BLOOMFILTERBITCOUNT) = 1;

//return \*(bloomfilter.dicthash + tmp % bloomfilter.BLOOMFILTERBITCOUNT) == 1;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称: Bloomclass::hash2(char\* buffer, int command = 0)

函数功能: 将字符串按照ascii码变为数字，并得到其和，最后对比特长度取余，如果int参数为1则将bloomfilter对应标志位置1，否则跳过。之后查询该位置是否为1.

传入参数: char\*目标字符串，int 0：查询，1：更改

返回值: 如果是1返回true，否则为false。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool Bloomclass::hash2(char\* buffer, int command = 0) {//Deprecated

unsigned int tmp = 0;

while (\*buffer != '\0')

{

tmp += (int)(\*buffer);

buffer++;

}

if (command == 1)

return turnbit(tmp % bloomfilter.BLOOMFILTERBITCOUNT);

return checkbit(tmp % bloomfilter.BLOOMFILTERBITCOUNT);

//\*(bloomfilter.dicthash + tmp % bloomfilter.BLOOMFILTERBITCOUNT) = 1;

//return \*(bloomfilter.dicthash + tmp % bloomfilter.BLOOMFILTERBITCOUNT) == 1;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称: Bloomclass::hash3(char\* buffer, int command = 0)

函数功能: 将字符串按照ascii码变为数字，并得到平方根取整的和，最后对比特长度取余，如果int参数为1则将bloomfilter对应标志位置1，否则跳过。之后查询该位置是否为1.

传入参数: char\*目标字符串，int 0：查询，1：更改

返回值: 如果是1返回true，否则为false。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool Bloomclass::hash3(char\* buffer, int command = 0) {//Deprecated

unsigned int tmp = 1;

while (\*buffer != '\0')

{

tmp \*= (int)(sqrt((double)\*buffer));

buffer++;

}

if (command == 1)

return turnbit(tmp % bloomfilter.BLOOMFILTERBITCOUNT);

return checkbit(tmp % bloomfilter.BLOOMFILTERBITCOUNT);

//\*(bloomfilter.dicthash + tmp % bloomfilter.BLOOMFILTERBITCOUNT) = 1;

//return \*(bloomfilter.dicthash + tmp % bloomfilter.BLOOMFILTERBITCOUNT) == 1;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称: Bloomclass::hash4(char\* buffer, int command = 0)

函数功能: 将字符串按照ascii码变为数字，并按+-+-。。。顺序依次加减，最后对比特长度取余，如果int参数为1则将bloomfilter对应标志位置1，否则跳过。之后查询该位置是否为1.

传入参数: char\*目标字符串，int 0：查询，1：更改

返回值: 如果是1返回true，否则为false。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool Bloomclass::hash4(char\* buffer, int command = 0) {//Deprecated

unsigned int tmp = 0;

unsigned int sign = 1;

while (\*buffer != '\0')

{

tmp += sign \* (int)(\*buffer);

sign \*= -1;

buffer++;

}

if (command == 1)

return turnbit(tmp % bloomfilter.BLOOMFILTERBITCOUNT);

return checkbit(tmp % bloomfilter.BLOOMFILTERBITCOUNT);

//\*(bloomfilter.dicthash + tmp % bloomfilter.BLOOMFILTERBITCOUNT) = 1;

//return \*(bloomfilter.dicthash + tmp % bloomfilter.BLOOMFILTERBITCOUNT) == 1;

}

// BKDR Hash Function

bool Bloomclass::hash5(char\* buffer, int command = 0) {

if (Bloomclass::gethashfunction\_num() < 1) {//当设置完散列函数个数后，可以用此进行顺序排除

return true;

}

unsigned int tmp = 0;

unsigned int seed = 131; // 31 131 1313 13131 131313 etc..

while (\*buffer)

{

tmp = tmp \* seed + (\*buffer++);

}

tmp = tmp & 0x7FFFFFFF;

if (command == 1)

return turnbit(tmp % bloomfilter.BLOOMFILTERBITCOUNT);

return checkbit(tmp % bloomfilter.BLOOMFILTERBITCOUNT);

}

// AP Hash Function

bool Bloomclass::hash6(char\* buffer, int command = 0) {

if (Bloomclass::gethashfunction\_num() < 2) {//当设置完散列函数个数后，可以用此进行顺序排除

return true;

}

unsigned int tmp = 0;

int i;

for (i = 0; \*buffer; i++){

if ((i & 1) == 0){

tmp ^= ((tmp << 7) ^ (\*buffer++) ^ (tmp >> 3));

}

else{

tmp ^= (~((tmp << 11) ^ (\*buffer++) ^ (tmp >> 5)));

}

}

tmp = tmp & 0x7FFFFFFF;

if (command == 1)

return turnbit(tmp % bloomfilter.BLOOMFILTERBITCOUNT);

return checkbit(tmp % bloomfilter.BLOOMFILTERBITCOUNT);

}

// DJB Hash Function

bool Bloomclass::hash7(char\* buffer, int command = 0) {

if (Bloomclass::gethashfunction\_num() < 3) {//当设置完散列函数个数后，可以用此进行顺序排除

return true;

}

unsigned int tmp = 5381;

while (\*buffer)

{

tmp += (tmp << 5) + (\*buffer++);

}

tmp = tmp & 0x7FFFFFFF;

if (command == 1)

return turnbit(tmp % bloomfilter.BLOOMFILTERBITCOUNT);

return checkbit(tmp % bloomfilter.BLOOMFILTERBITCOUNT);

}

//JSHash Function

bool Bloomclass::hash8(char\* buffer, int command = 0) {

if (Bloomclass::gethashfunction\_num() < 4) {//当设置完散列函数个数后，可以用此进行顺序排除

return true;

}

unsigned int tmp = 1315423911;

while (\*buffer)

{

tmp ^= ((tmp << 5) + (\*buffer++) + (tmp >> 2));

}

tmp = tmp & 0x7FFFFFFF;

if (command == 1)

return turnbit(tmp % bloomfilter.BLOOMFILTERBITCOUNT);

return checkbit(tmp % bloomfilter.BLOOMFILTERBITCOUNT);

}

// RS Hash Function

bool Bloomclass::hash9(char\* buffer, int command = 0) {

if (Bloomclass::gethashfunction\_num() < 5) {//当设置完散列函数个数后，可以用此进行顺序排除

return true;

}

unsigned int tmp = 0;

unsigned int b = 378551;

unsigned int a = 63689;

while (\*buffer)

{

tmp = tmp \* a + (\*buffer++);

a \*= b;

}

tmp = tmp & 0x7FFFFFFF;

if (command == 1)

return turnbit(tmp % bloomfilter.BLOOMFILTERBITCOUNT);

return checkbit(tmp % bloomfilter.BLOOMFILTERBITCOUNT);

}

//SDBM hash function

bool Bloomclass::hash10(char\* buffer, int command = 0) {

if (Bloomclass::gethashfunction\_num() < 6) {//当设置完散列函数个数后，可以用此进行顺序排除

return true;

}

unsigned int tmp = 0;

while (\*buffer)

{

// equivalent to: hash = 65599\*hash + (\*str++);

tmp = (\*buffer++) + (tmp << 6) + (tmp << 16) - tmp;

}

tmp = tmp & 0x7FFFFFFF;

if (command == 1)

return turnbit(tmp % bloomfilter.BLOOMFILTERBITCOUNT);

return checkbit(tmp % bloomfilter.BLOOMFILTERBITCOUNT);

}

//FNVHash function

bool Bloomclass::hash11(char\* buffer, int command = 0) {

if (Bloomclass::gethashfunction\_num() < 7) {//当设置完散列函数个数后，可以用此进行顺序排除

return true;

}

unsigned int tmp = 0;

int fnvprime = 0x811C9DC5;

while (\*buffer) {

tmp \*= fnvprime;

tmp ^= (int)(\*buffer++);

}

tmp = tmp & 0x7FFFFFFF;

if (command == 1)

return turnbit(tmp % bloomfilter.BLOOMFILTERBITCOUNT);

return checkbit(tmp % bloomfilter.BLOOMFILTERBITCOUNT);

}

//JAVA string hash function

bool Bloomclass::hash12(char\* buffer, int command = 0) {

if (Bloomclass::gethashfunction\_num() < 8) {//当设置完散列函数个数后，可以用此进行顺序排除

return true;

}

unsigned int tmp = 0;

while (\*buffer) {

tmp = tmp \* 31 + (\*buffer++);

}

tmp = tmp & 0x7FFFFFFF;

if (command == 1)

return turnbit(tmp % bloomfilter.BLOOMFILTERBITCOUNT);

return checkbit(tmp % bloomfilter.BLOOMFILTERBITCOUNT);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称: Bloomclass::AddString(char\* buffer)

函数功能: 添加目标字符串到bloomfilter的特征位为1

传入参数: char\*目标字符串

返回值: 如果符合返回true，否则为false。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool Bloomclass::AddString(char\* buffer)

{

return hash5(buffer, 1) && hash6(buffer, 1) && hash7(buffer, 1) && hash8(buffer, 1) &&

hash9(buffer, 1) && hash10(buffer, 1) && hash11(buffer, 1) && hash12(buffer, 1);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称: Bloomclass::checkstring(char\* buffer)

函数功能: 检查目标字符串是否在bloomfilter的特征位为1

传入参数: char\*目标字符串

返回值: 如果符合返回true，否则为false。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool Bloomclass::checkstring(char\* buffer)

{

if (Bloomclass::excludeString(buffer) || buffer==NULL || \*buffer=='\0') {

return false;

}

strcpy\_s(llllltmpbuffer, lllltmpbuffer);

strcpy\_s(lllltmpbuffer, llltmpbuffer);

strcpy\_s(llltmpbuffer, lltmpbuffer);

strcpy\_s(lltmpbuffer, ltmpbuffer);

strcpy\_s(ltmpbuffer, tmpbuffer);

strcpy\_s(tmpbuffer , buffer);

return hash5(buffer, 0) && hash6(buffer, 0) && hash7(buffer, 0) && hash8(buffer, 0) &&

hash9(buffer, 0) && hash10(buffer, 0) && hash11(buffer, 0) && hash12(buffer, 0);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称: Bloomclass::exportdict(TCHAR\* path)

函数功能: 将bloomfilter结构体的二进制位数据导出到目标路径

传入参数: TCHAR\*文件路径

返回值: 如果操作完成返回的bloomfilter的二进制位的个数，否则为-1。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int Bloomclass::exportdict(TCHAR\* GLOBALDICTFILENAME)

{

FILE\* fp;

\_wfopen\_s(&fp, GLOBALDICTFILENAME, L"w+");

if (fp != NULL)

{

char checklegal[] = "bfd";

fwrite(checklegal, sizeof(char), 4, fp);

fwrite(&(bloomfilter.BLOOMFILTERBITCOUNT), sizeof(int), 1, fp);

fwrite(bloomfilter.dicthash, sizeof(unsigned short), bloomfilter.allocatesize, fp);

fclose(fp);

return bloomfilter.BLOOMFILTERBITCOUNT;

}

return -1;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称: Bloomclass::importdict(TCHAR\* path)

函数功能: 向bloomfilter结构体导入来自指定路径文件内的二进制位数据，这个文件的开头以“bfd”开头，之后以一个int表示所占用的二进制位的个数

传入参数: TCHAR\*类型的路径

返回值: 如果操作完成返回0，否则为-1。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int Bloomclass::importdict(TCHAR\* path)

{

FILE\* fp;

int ret = \_wfopen\_s(&fp, path, \_T("r"));

if (ret == 2)

{

MessageBox(NULL, L"NO specific dict file found!", \_T("Error"), NULL);

return -1;

}

if (fp == NULL)

{

//wprintf(publicbuffer, );

MessageBox(NULL, L"No dict file found", \_T("Error"), NULL);

return -1;

}

rewind(fp);

char checklegal[4] = { 0 };

fread(checklegal, sizeof(char), 4, fp);

if (strcmp(checklegal, "bfd"))

{

MessageBox(NULL, L"Please choose the dict file.", L"Error", NULL);

return -1;

}

fread(&(bloomfilter.BLOOMFILTERBITCOUNT), sizeof(int), 1, fp);

initspellchecker(bloomfilter.BLOOMFILTERBITCOUNT);

fread(bloomfilter.dicthash, sizeof(unsigned short), bloomfilter.allocatesize, fp);

fclose(fp);

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称: Bloomclass::releasedict(int bitcount)

函数功能: 释放bloomfilter结构体所占用的内存

传入参数: 无

返回值: 如果操作完成返回0，否则不为0。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int Bloomclass::releasedict()

{

free(bloomfilter.dicthash);

bloomfilter.dicthash = NULL;

bloomfilter.BLOOMFILTERBITCOUNT = 0;

bloomfilter.allocatesize = 0;

return bloomfilter.BLOOMFILTERBITCOUNT;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称: Bloomclass::initspellchecker(int bitcount)

函数功能: 初始化bloomfilter结构体

传入参数: 需要的bloomfilter的二进制位个数，默认64

返回值: 操作成功返回申请到的bloomfilter的二进制位个数。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int Bloomclass::initspellchecker(int bitcount = 64)

{

bloomfilter.BLOOMFILTERBITCOUNT = bitcount;

bloomfilter.allocatesize = bloomfilter.BLOOMFILTERBITCOUNT / (sizeof(unsigned short)\*8) + 1;

bloomfilter.dicthash = (unsigned short\*)malloc(bloomfilter.allocatesize \* sizeof(unsigned short));

int i;

for (i = 0; i < bloomfilter.allocatesize; i++)

{

\*(bloomfilter.dicthash + i) = 0;

}

return bloomfilter.BLOOMFILTERBITCOUNT;

}

int Bloomclass::GetBloomlen() {

return bloomfilter.BLOOMFILTERBITCOUNT;

}

int Bloomclass::SetBloomlen(int len) {

Bloomclass::releasedict();

return Bloomclass::initspellchecker(len);

}

bool Bloomclass::checkbit(int position) {

if (position < bloomfilter.BLOOMFILTERBITCOUNT) {

unsigned short comparepart = \*(bloomfilter.dicthash + position / (sizeof(unsigned short) \* 8));

unsigned short comparepart\_offset = position % (sizeof(unsigned short) \* 8);

comparepart >>= 4 \* (3 - comparepart\_offset / 4);

comparepart &= 15/\*1111\*/;

return comparepart & (unsigned short)pow(2, 3 - comparepart\_offset % 4);

}

else {

printf("Error in Bloomclass::checkbit");

}

}

bool Bloomclass::turnbit(int position) {

if (position < bloomfilter.BLOOMFILTERBITCOUNT) {

unsigned short comparepart = \*(bloomfilter.dicthash + position / (sizeof(unsigned short) \* 8));

unsigned short comparepart\_offset = position % (sizeof(unsigned short) \* 8);

comparepart >>= 4 \* (3 - comparepart\_offset / 4);

comparepart &= 15/\*1111\*/;

comparepart |= (unsigned short)pow(2, 3 - comparepart\_offset % 4);

comparepart <<= 4 \* (3 - comparepart\_offset / 4);

\*(bloomfilter.dicthash + position / (sizeof(unsigned short) \* 8)) &= (unsigned short)(65535 - 15\*pow(2, 4\*(3 - comparepart\_offset / 4)));

\*(bloomfilter.dicthash + position / (sizeof(unsigned short) \* 8)) |= comparepart;

return Bloomclass::checkbit(position);

}

else {

printf("Error in Bloomclass::turnbit");

}

}

void Bloomclass::echodicthash() {

for (int i = 0; i < bloomfilter.allocatesize; i++) {

printf("%d ", \*(bloomfilter.dicthash + i));

int ret = \*(bloomfilter.dicthash + i);

}

}

bool Bloomclass::excludeString(char\* buffer) {

return false;

}

bool Bloomclass::turnbit\_v1(int position) {//Deprecated

if (position < bloomfilter.BLOOMFILTERBITCOUNT) {

unsigned short comparepart = \*(bloomfilter.dicthash + position / (sizeof(unsigned short) \* 8));

unsigned short comparepart\_offset = position % (sizeof(unsigned short) \* 8);

switch (comparepart\_offset / 4) {

case 0: {

comparepart >>= 12;

//comparepart &= 15/\*1111\*/;

switch (comparepart\_offset % 4) {

case 0:comparepart |= 8/\*1000\*/; break;

case 1:comparepart |= 4/\*0100\*/; break;

case 2:comparepart |= 2/\*0010\*/; break;

case 3:comparepart |= 1/\*0001\*/; break;

default:printf("warning: turnbit1");

}

comparepart <<= 12;

\*(bloomfilter.dicthash + position / (sizeof(unsigned short) \* 8)) &= 4095/\*0000 1111 1111 1111\*/;

\*(bloomfilter.dicthash + position / (sizeof(unsigned short) \* 8)) |= comparepart;

break;

}

case 1: {

comparepart >>= 8;

comparepart &= 15/\*0000 1111\*/;

switch (comparepart\_offset % 4) {

case 0:comparepart |= 8/\*1000\*/; break;

case 1:comparepart |= 4/\*0100\*/; break;

case 2:comparepart |= 2/\*0010\*/; break;

case 3:comparepart |= 1/\*0001\*/; break;

default:printf("warning: turnbit2");

}

comparepart <<= 8;

\*(bloomfilter.dicthash + position / (sizeof(unsigned short) \* 8)) &= 61695/\*1111 0000 1111 1111\*/;

\*(bloomfilter.dicthash + position / (sizeof(unsigned short) \* 8)) |= comparepart;

break;

}

case 2: {

comparepart >>= 4;

comparepart &= 15/\*0000 0000 1111\*/;

switch (comparepart\_offset % 4) {

case 0:comparepart |= 8/\*1000\*/; break;

case 1:comparepart |= 4/\*0100\*/; break;

case 2:comparepart |= 2/\*0010\*/; break;

case 3:comparepart |= 1/\*0001\*/; break;

default:printf("warning: turnbit3");

}

comparepart <<= 4;

\*(bloomfilter.dicthash + position / (sizeof(unsigned short) \* 8)) &= 65295/\*1111 1111 0000 1111\*/;

\*(bloomfilter.dicthash + position / (sizeof(unsigned short) \* 8)) |= comparepart;

break;

}

case 3: {

//comparepart >>= 0;

comparepart &= 15/\*0000 0000 0000 1111\*/;

switch (comparepart\_offset % 4) {

case 0:comparepart |= 8/\*1000\*/; break;

case 1:comparepart |= 4/\*0100\*/; break;

case 2:comparepart |= 2/\*0010\*/; break;

case 3:comparepart |= 1/\*0001\*/; break;

default:printf("warning: turnbit4");

}

//comparepart <<= 0;

\*(bloomfilter.dicthash + position / (sizeof(unsigned short) \* 8)) &= 65520/\*1111 1111 1111 0000\*/;

\*(bloomfilter.dicthash + position / (sizeof(unsigned short) \* 8)) |= comparepart;

break;

}

default:printf("warning: offset turnbit05"); return false;

}

//Bloomclass::echodicthash();

return Bloomclass::checkbit(position);

}

}

bool Bloomclass::checkbit\_v1(int position) {//Deprecated

if (position < bloomfilter.BLOOMFILTERBITCOUNT) {

unsigned short comparepart = \*(bloomfilter.dicthash + position / (sizeof(unsigned short) \* 8));

unsigned short comparepart\_offset = position % (sizeof(unsigned short) \* 8);

switch (comparepart\_offset / 4) {

case 0: {

comparepart >>= 12;

//comparepart &= 15/\*1111\*/;

switch (comparepart\_offset % 4) {

case 0:return comparepart & 8/\*1000\*/;

case 1:return comparepart & 4/\*0100\*/;

case 2:return comparepart & 2/\*0010\*/;

case 3:return comparepart & 1/\*0001\*/;

default:printf("warning: offset dealbit01");

}

break;

}

case 1: {

comparepart >>= 8;

comparepart &= 15/\*0000 1111\*/;

switch (comparepart\_offset % 4) {

case 0:return comparepart & 8/\*1000\*/;

case 1:return comparepart & 4/\*0100\*/;

case 2:return comparepart & 2/\*0010\*/;

case 3:return comparepart & 1/\*0001\*/;

default:printf("warning: offset dealbit02");

}

break;

}

case 2: {

comparepart >>= 4;

comparepart &= 15/\*0000 0000 1111\*/;

switch (comparepart\_offset % 4) {

case 0:return comparepart & 8/\*1000\*/;

case 1:return comparepart & 4/\*0100\*/;

case 2:return comparepart & 2/\*0010\*/;

case 3:return comparepart & 1/\*0001\*/;

default:printf("warning: offset dealbit03");

}

break;

}

case 3: {

//comparepart >>= 0;

comparepart &= 15/\*0000 0000 0000 1111\*/;

switch (comparepart\_offset % 4) {

case 0:return comparepart & 8/\*1000\*/;

case 1:return comparepart & 4/\*0100\*/;

case 2:return comparepart & 2/\*0010\*/;

case 3:return comparepart & 1/\*0001\*/;

default:printf("warning: offset dealbit04");

}

break;

}

default:printf("warning: offset dealbit05"); return false;

}

}

}

Bloomclass::Bloomclass(int income) {

initspellchecker(income);

}

Bloomclass::~Bloomclass() {

releasedict();

}

###### 2. 完整工程文件

****

###### 3. 测试数据

1. 测试布隆过滤器文件



1. 测试代码文件



###### 4. 运行结果

