**课程编号：0521172B** **课程性质：必修**

**数据结构课程设计报告**

**学院： 合肥工业大学**

**专业： 物联网工程**

**班级： 17-2班**

**姓名： 刁肥宅**

**学号： 201721XXXX**

**教师： 周波**

**2018年 8月 10日 至 2018年 8 月 16日**

# 第一章 理论基础

## 1.1 课题背景

在学习和工作中，时常需要使用到文本编辑器进行文档或者代码的编写。而文档和代码的编写中，常常会出现类似”true”和”ture”，”main”和”mian”等等的混淆，在程序排查错误和文档阅读中，造成不小的麻烦。因此，常用的文本编辑器例如Microsoft Word，Microsoft Visual Studio Code，Code::block等等通常都会附带有拼写检查和关键词高亮的功能。其原理是，通过引入默认的词典，记录该词典中的所有字符串，和当前文档进行比对，从而发现错误的代词或者代码关键字，并且采用下划线，文本高亮等形式来提醒用户。甚至于在用户输入某个关键词的前几个字符时，自动提示补全关键字。

然而，如果直接记录词典中的所有字符串并且用以遍历比对，会极大程度地加大空间和时间的复杂度。因此，在拼写检查器UNIXspell-checkers中，就采用了布隆过滤器（Bloom Filter, BF）数据结构的哈希表来判断某个字符串是否存在于词典中。

因此，本课程设计将使用Qt来设计文本编辑器，并建立以布隆过滤器作为词典存储的数据结构，以此实现拼写检查的功能。同时，该软件将支持添加关键字功能，以满足用户多变的需求。

本文将以如下顺序进行阐述：第一章将介绍该课程设计的理论基础，包括数据存储结构原理，程序设计框架等，第二章将介绍该课程设计的程序细节以及效果展示，第三章则将讨论和分析本次试验结果。

## 1.2 布隆过滤器

布隆过滤器是在1970年被巴顿布隆提出的，用来解决数据存储问题的哈希表。它将数据的存储形式由诸多字符串组成的数组，变为一个二进制向量序列，极大减小了检索和存储的难度。哈希表又称为散列表，是实现字典操作的一种有效数据结构。尽管在最坏情况下，散列表中查找一个元素的时间与在链表中查找的时间相同，达到了Θ(n)。然而在实际应用中，散列表查找的性能是极好的。在一些合理的假设下，在散列表中查找一个元素的平均时间是O(1)。

布隆过滤器的原理如下：初始化一个n字节二进制向量Array全部置为0，对于每个希望存储的字符串str1，通过x个哈希函数（又称散列函数）获取取值在[0, n)的哈希值序列H，然后依次将Array中的第Hi个字节置为1，则完成了对该字符串str1的存储。因此，同样地，对于希望检索的字符串str2，通过相同的哈希函数获取得到哈希值序列H’，显然，当该序列已被存储时，该哈希值序列在Array中对应的每一个字节都应该为1。图1.1展示了该结构的原理。

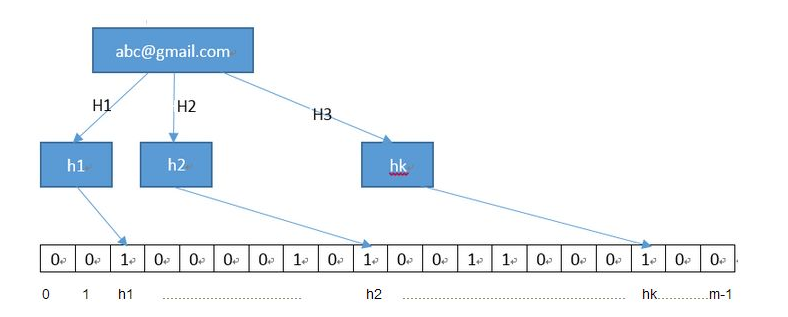


图1.1 布隆过滤器原理

所以，布隆过滤器的索引时间复杂度只和散列函数个数有关而与数据存储量大小无关；散列函数本身也彼此独立，利于并行计算，因而在各种数据结构之中脱颖而出。而它的缺点也同样地明显，首先，它不能删除已存储的元素。其次，对于某些本未被存储的字符串和已存储若干个字符串共m个字节的二进制向量，可能会存在一定概率导致未被存储的字符串被误判为已存储。

因此，有两种方法来尽量减小误判问题带来的影响：第一，减小误判率p，即使用较大容量的Array从而提高n,使用较多个哈希函数从而提高x，避免在同一个布隆过滤器中存储过多的字符串从而减小x。然而，这些方法都会一定程度上牺牲字符串存储的效率；第二，建立布隆过滤器白名单，对于那些未被存储而被误判为已被存储的字符串，加入到该白名单中，然而该白名单会对于布隆过滤器的检索，添加功能造成很多麻烦。因此，这两种方法都不能从根本上消除布隆过滤器的误判率。

尽管如此，布隆过滤器仍是一个相对而言，高效而鲁棒的数据存储结构，也是本次课程设计的研究重点。

## 1.3 Qt

Qt是一个跨平台C++图形用户界面应用程序开发框架，可以非常方便地用于开发GUI程序。本次课程设计将Qt用于文本编辑器的开发，并采用Qt Creator作为程序开发环境。

程序主要使用了QTextEdit控件以满足复制、粘贴、剪切、删除、撤销等文本编辑器基本功能，和QSyntaxHighlighter库以对满足规则的文本对象进行高亮。

# 第二章 程序设计

## 2.1 文本编辑器设计

本程序使用QTextEdit控件设计文本编辑器，并继承QSyntaxHighlighter类编写子类myHighLight类以实现高亮功能。

用户通过QTextEdit控件输入文本，并将文本传至myHighLight类的构造函数用以初始化。在该构造函数中，同时初始化了QRegExp\*类型变量rule以匹配关键字用并使用QTextCharFormat类型变量format高亮。文本编辑器的界面如图2.2所示。

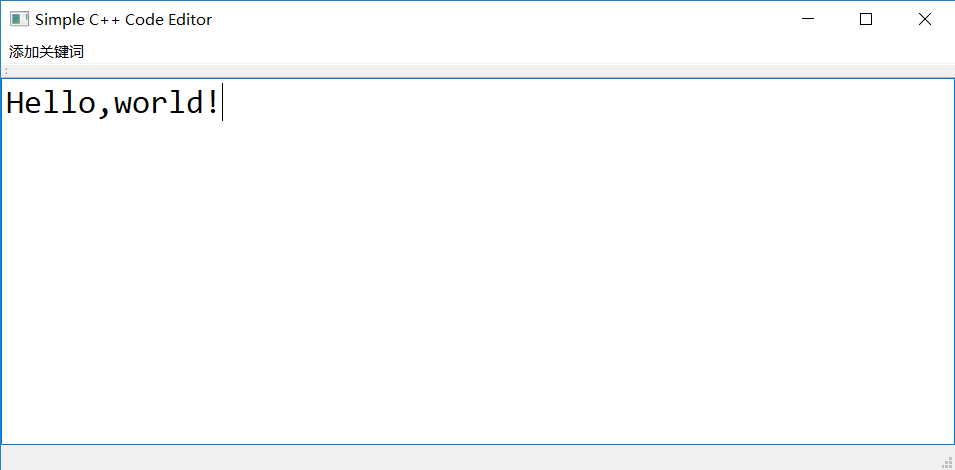


图2.2 文本编辑器

值得一提的是，该文本编辑器同时也支持了复制、粘贴、剪切、删除、撤销等等文本操作功能。

## 2.2 布隆过滤器设计

布隆过滤器设计的关键主要在两方面：第一，布隆过滤器的参数如散列函数个数x，二进制数组比特位数n等等；第二，散列函数的设计。

为了将误判率控制在1%以下，取散列函数个数x = 7，而比特位数n = 10\*m,其中m为预计希望保存的关键字个数（具体推导过程见3.1节）。该程序内置了近百个C/C++关键字，经过一定的建模分析后，预计希望最终保存的关键字为8000个。因此，n = 80000字节，以大小为10000的unsigned char类型数组形式保存。

理论上优秀的散列函数应该包含下列性质： 1.哈希得到的值均匀分布在0-n上；2.各个哈希函数之间彼此独立，各个字符串得到的哈希值彼此独立。因此该程序的哈希函数设计宗旨是，1.获得尽量大的数值后取余，从而确保哈希值能够均匀分布。2.各个哈希函数之间无耦合关系，从而保证独立性。

因此，在myHighLight类被构造的同时，开辟一个大小为10000的unsigned char类型数组内存空间，并全部初始化为0.

## 2.3 添加关键词功能设计

对布隆过滤器的操作功能包括查找关键字isKeyExisted和添加关键字appendKey两个部分。

对关键字的查找即使判断该字符串在散列函数中获得的每个哈希值hash是否满足二进制阵列BloomFilter的第hash个字节位置处是否为1，即，BloomFilter[hash/8] >> hash%8) % 2 == 1，当任一哈希值不满足时，则认为该关键字不存在。而在文本编辑器中，首先使用正则表达式rule提取出文本中的所有单词，而仅当该单词被存储为关键字时，将该段文本高亮。

添加关键字之前首先应该使用正则表达式判断该段文本是否为单词，然后判断该关键字是否已经存在，虽然将该字符串在散列函数中获得的每个哈希值hash处对应的二进制阵列BloomFilter的第hash个字节位置处的所有0置为1，即 BloomFilter[code/8] += 1 << code%8。使用该方法，程序内置了C++17的所有关键字，如表2.1所示。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| asm | do | if | return | typedef |
| auto | double | inline | short | typeid |
| bool | dynamic\_cast | int | signed | typename |
| break | else | long | sizeof | union |
| case | enum | mutable | static | unsigned |
| catch | explicit | namespace | static\_cast | using |
| char | export | new | struct | virtual |
| class | extern | operator | switch | void |
| const | false | private | template | volatile |
| const\_cast | float | protected | this | wchar\_t |
| continue | for | public | throw | while |
| default | friend | register | true |  |
| delete | goto | reinterpret\_cast | try |  |

表2.1 程序内置的所有关键字

在此基础上，该文本编辑器就具有了一定的编辑c++代码功能，效果如图2.3所示。

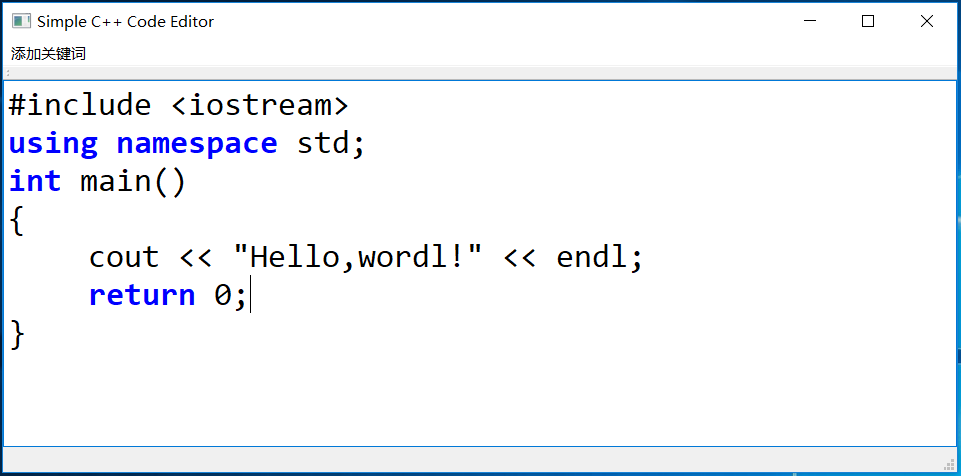


图2.3 程序效果展示

程序可以添加选定内容为关键字，从而一劳永逸地高亮该文本，人为添加关键词后，应自动重新对整段文本进行关键词检测。如图2.4所示。

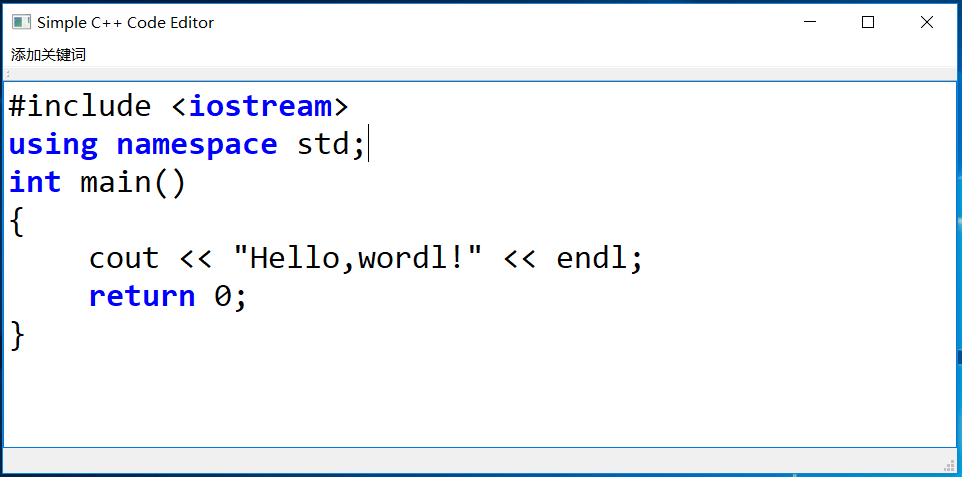


图2.4 添加关键字

# 第三章 结果分析

## 3.1 布隆过滤器的鲁棒性分析

布隆过滤器的鲁棒性分析即对布隆过滤器的误判率进行分析。正如前文所说，布隆过滤器的误判率不可避免，因此只能设置阈值1%，通过参数的设计，将布隆过滤器的理论误判概率降低到阈值以下。

首先假设哈希函数为理论完美的，即哈希后得到的值符合均匀分布，那么每个字符串的每个哈希值都会等概率地生成在二进制数组的n个比特位上，并且彼此独立。那么，基于此假设，插入单个哈希值时，一个特定比特位没有被激活，即置为1，的概率为p0 = 1 – 1/n。字符串对应了x个哈希值，由于哈希值之间彼此独立，则在插入单个字符串后，该比特位没有被激活的概率为p1 = p0x。而插入k个字符串后，同理该比特位没有被激活的概率为p2 = p1k，反之它被激活的概率就为p2’ = 1 - p2。而误判现象即对于一个未被存储的元素，x个对应的比特位均被激活的概率，故误判率为P = (p2’)k，由于1/m趋近于0，故P可以简化为P = （1 – e-kx/n）k。在给定的x和n下，最小化P得到k = (n/x)ln2，此时ln P = -(n/x)ln22。对于P<0.01，有n/x>9.6，而k = 6.72。因而，在本程序的布隆过滤器中，取哈希函数共7个，二进制数组字节数为80000。

## 3.2 运行复杂度分析

布隆过滤器作为数据结构时，添加和查找关键词的运算复杂度极低，而相应的删除关键词的成本相当高。

布隆过滤器删除关键词通常采用引入白名单的方式，假设白名单共m个字符串，则遍历白名单的复杂度约为O(m\*s)，其中s为字符串平均长度。由于查找关键词前必须遍历白名单，因此复杂度为O(x)+O(m\*s)，其中x为散列函数个数；而添加关键词前必须要先查找关键词，故运算复杂度为O(2\*x)+O(m\*x)。同样地，布隆过滤器的空间复杂度具有类似的情况，布隆过滤器对内存的占用分为两部分，第一，白名单存储，复杂度约为O(m\*s)；第二，二进制数组存储，复杂度约为O(n)。合计为O(m\*s)+ O(n)