（报告封面）



“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**  邹 鹏

**学 号**  2018218799

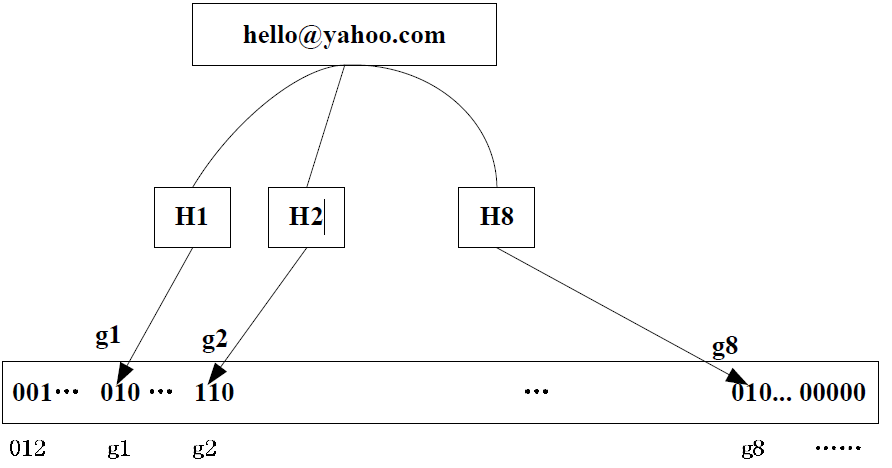
**专 业**  计算机科学与技术

**班 级**  计科18-2班

**完成日期**  2020/5/1

**（一） 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。

**（二） 设计要求**

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。

**（三） 实现描述**

① 通过QT实现简单的文本编辑器，MainWindow类是窗口，添加 new、open、save as、copy等等一些列控件操作，在mainwindow.cpp实现类成员函数on\_actionExit\_triggered()、on\_actionCopy\_triggered()等即可。

② 写一个文本高亮类MyHighlighter，对窗口的文本编辑控件的内容进行语法高亮。即mainwindow.cpp的构造函数里的highlighter = new MyHighlighter(ui->textEdit->document())语句。该类的声明与实现分别在在myhighlighter.h与myhighlighter.cpp中，此类继承QSyntaxHighlighter，根据QT官方文档的描述(<https://doc.qt.io/qt-5/qsyntaxhighlighter.html#details>)。继承这个类之后，在执行上述语句时，会自动调用我们自己编写的highlightBlock()成员函数。也即我们的整个语法高亮的实现函数。我们可以在此类中对输入的文本进行判断是否高亮。

③ 由上述所描述。在highlighterBlock中我们用isKeyExist()对文本进行判断。这个函数调用BloomFilter::KeyExistInBitFilter(const std::string &str)。也即我们用布隆过滤器的数据结构判断是否为关键字。

④ BlommFilter::KeyExistInBitFilter则通过调用哈希函数，对文本进行分析计算，判断计算得到的每一个bit是否在bitFilter中。

⑤ 我们的哈希函数一共采用了三个哈希函数，分别是BKDHash、RSHash、APHash，可对输入文本进行快速分析是否为关键字。

下面是各文件所承担功能：



**（三） 用户手册**

运行可执行文件时：

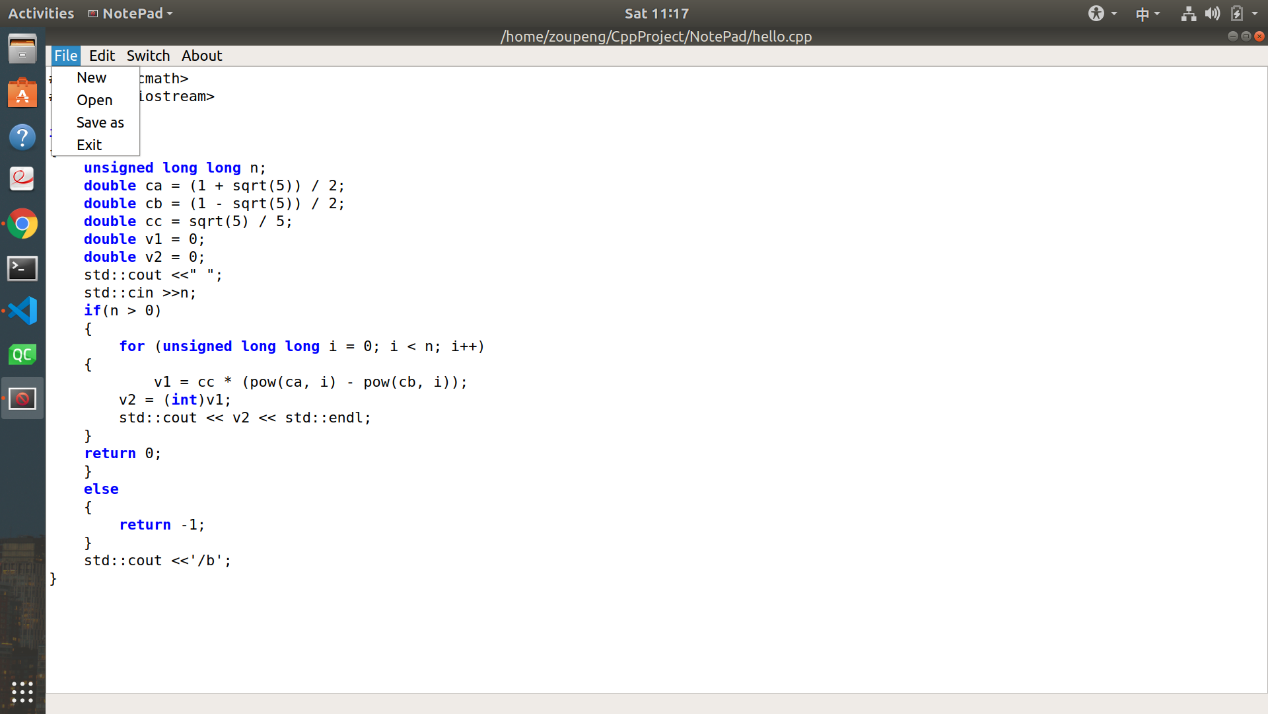
1. 可点击File进行打开或者新建文件，编辑完成后可以选择保存后退出。
2. 可点击Edit进行文件的拷贝、粘贴、剪切、撤销、重做功能。
3. Switch是切换高亮语言，目前可供选择的高亮语言关键字有C++、Java、Python，底层高亮实现采用布隆过滤器的数据结构。
4. 在输入过程中，会对输入文本进行判断，是否为该语言的关键字，并进行语法高亮。

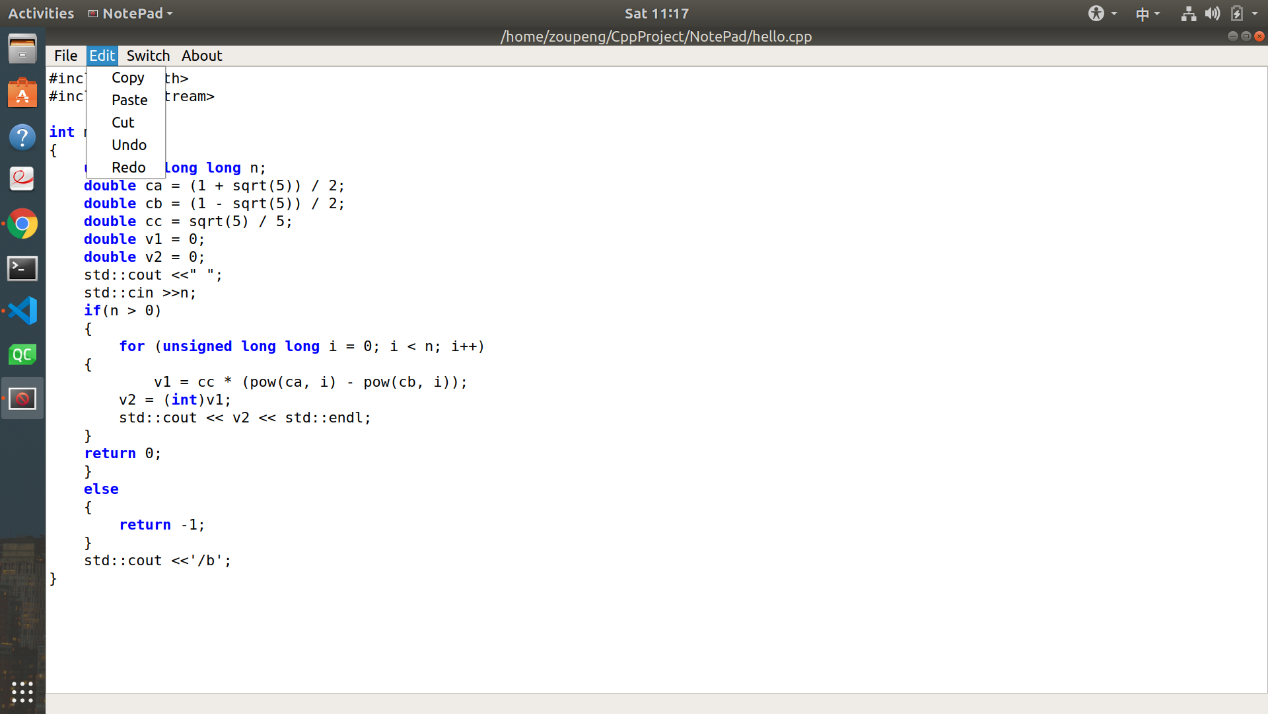
**（四） 调试及测试**

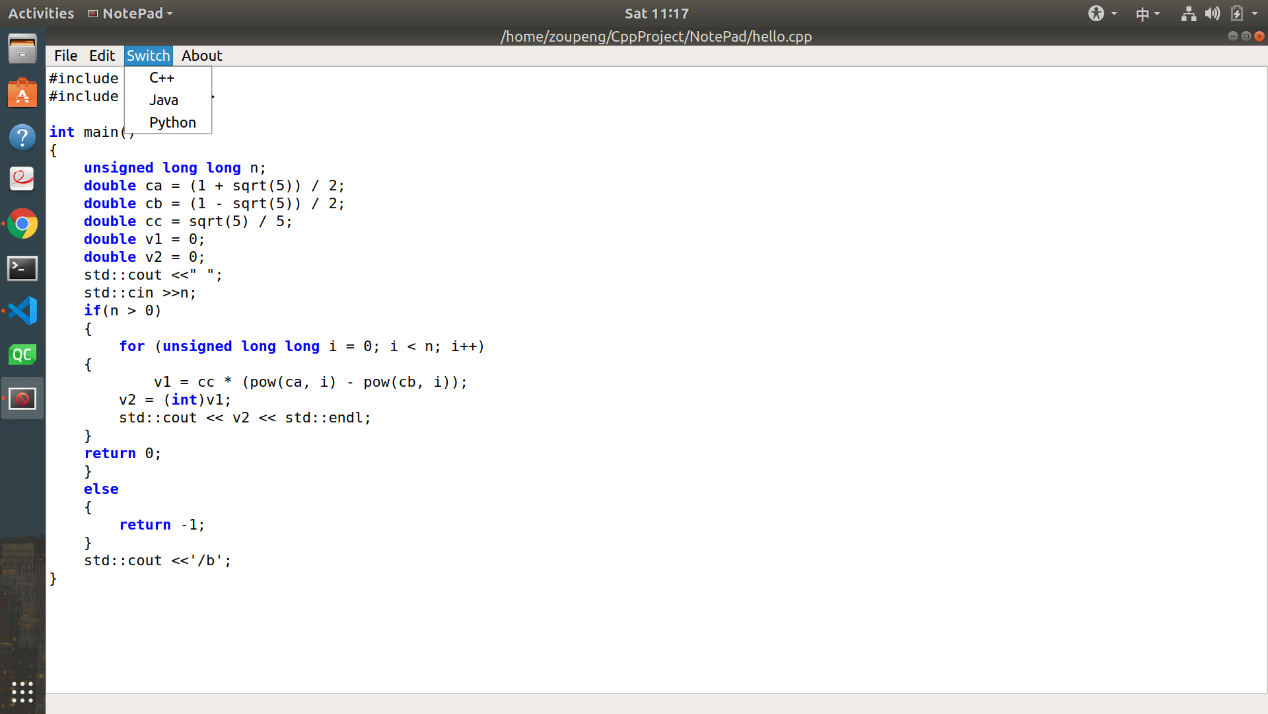
点击Switch进行切换输入，分别打开准备好的hello.cpp、hello.java、hello.py，点击Switch进行语法高亮语言选择，发现对应关键字会高亮，输入新的关键字也能够实现高亮，同时这里对c++、java增加了注释高亮功能。

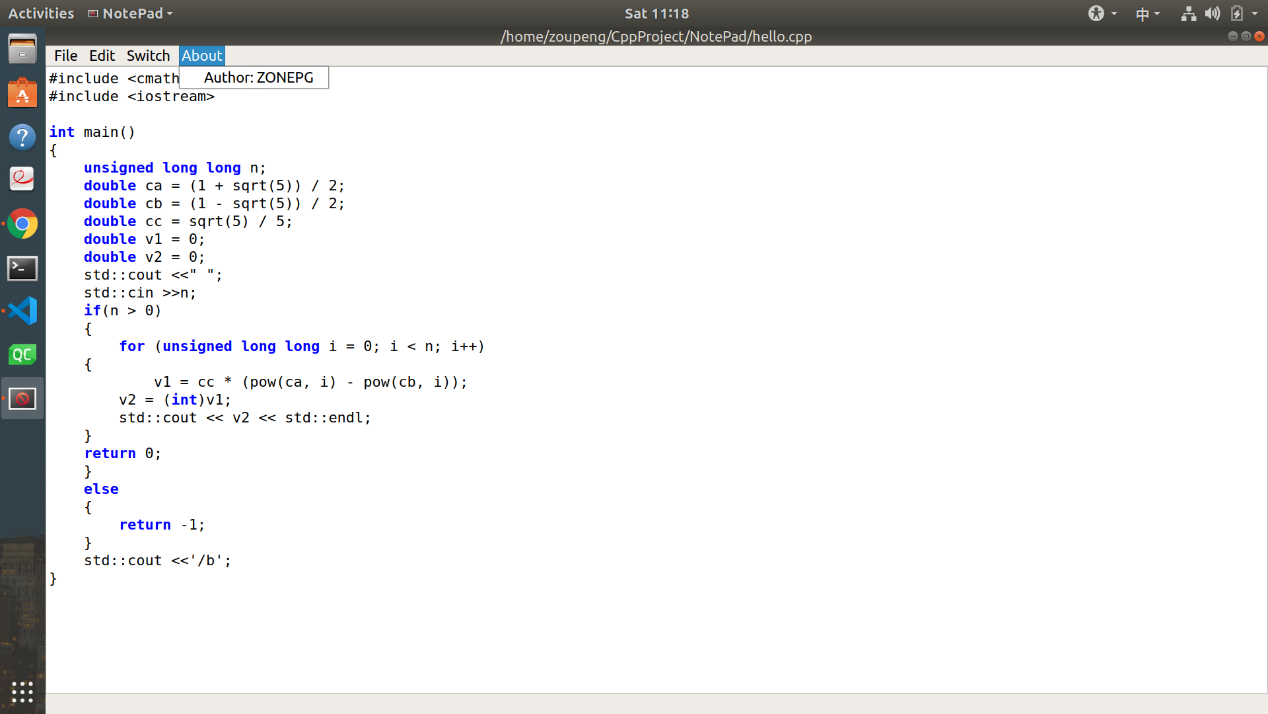
**（五） 运行实例：**

**① 选择打开hello.cpp，默认为c++关键字语法高亮**

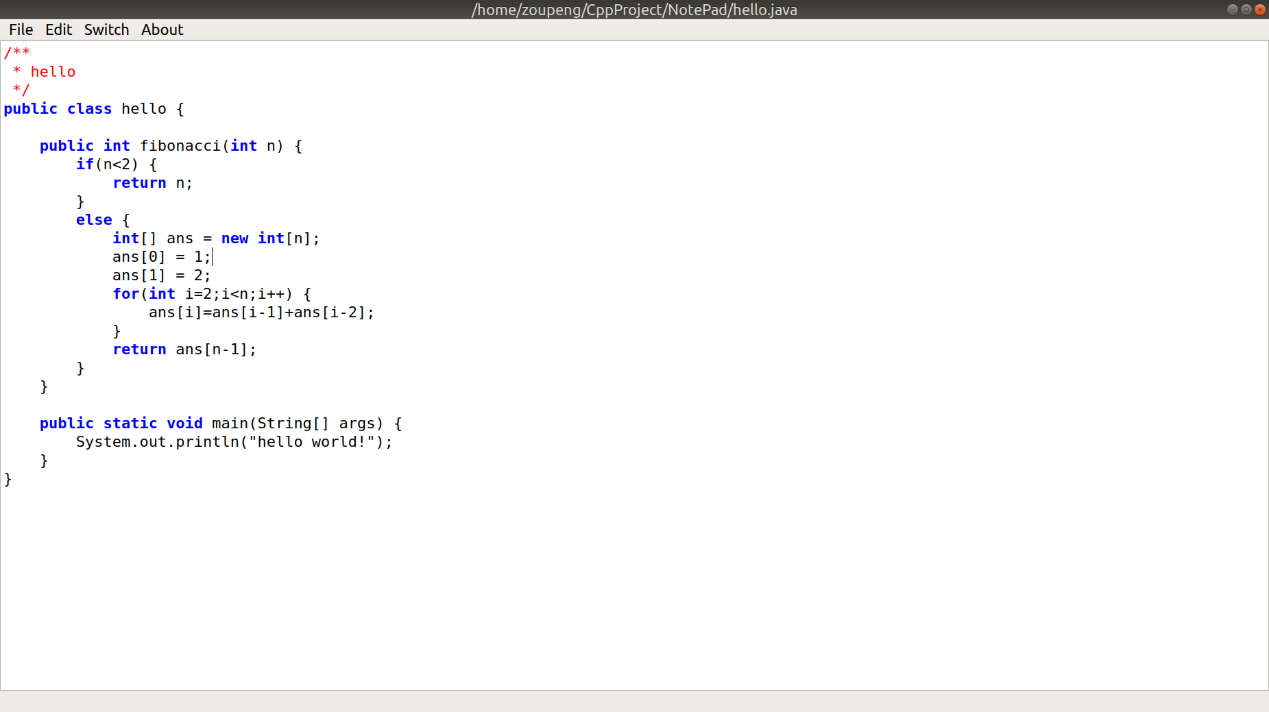




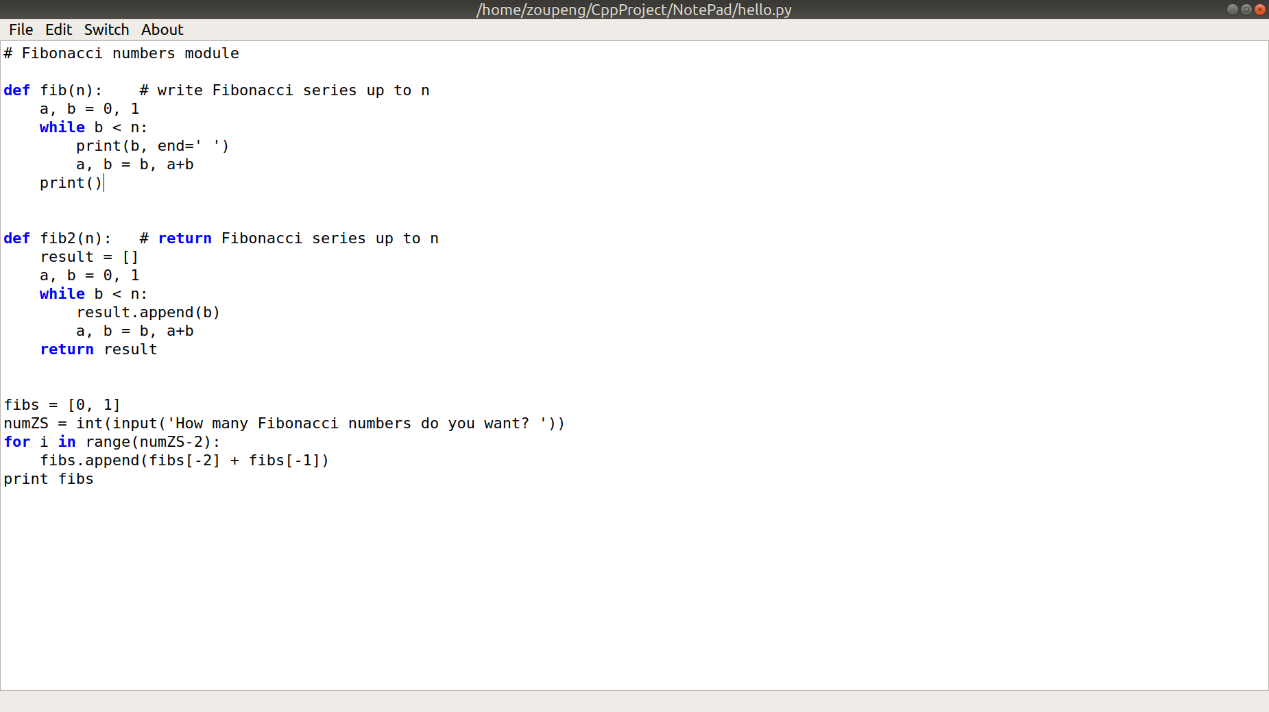




**② 选择打开hello.java，点击Switch切换java关键字语法高亮。**



**③ 选择打开hello.py，点击Swtich切换python关键字语法高亮。**



**（六）进一步改进**

程序已经实现的功能包括：

1. 文本编辑器的基本功能。
2. C++、java、Python关键字语法高亮设置
3. 布隆过滤器使用了3各哈希函数进行处理。
4. bitFilter的大小为870个bit/char。

可进一步改进的地方：

1. 如果想要加入新的语言关键字语法高亮可以自行添加xxx.txt文件，文件存入该语言的所有关键字。这里我们存储语言关键字是通过程序读文件的方式运行的。
2. 注释高亮部分只实现了/\* \*/的注释设置，其他注释高亮设置需要重新添加注释功能。
3. 布隆过滤器的数据结构采用了3个Hash函数与大小为870bit的bitFilter，在作者的测试中，上述三种语言的关键字高亮均已通过，不过对于处理文本较大、关键字较大的文件，计算速度可能会变慢。

**（七）心得体会**

（可描述课程设计过程中出现的主要问题、原因及解决方法，课程设计的主要收获等）

在实现本次课程设计过程中，我遇到以下问题较为棘手，在解决过程中花费了不少时间：

1. 选择语法高亮库，选择GUI界面 交互的库为QT，阅读了许多QT的官方英文文档，并仔细阅读其功能源码，才利用到自己的项目中。
2. 设计布隆过滤器的数据结构时，纠结了几种设计模式，最终选择了在 布隆过滤器的BloomFilter类中提供条件判断接口，供高亮类调用。
3. 读入关键字时，采用了将关键字txt文本读入数组中，然后用此数组构建布隆过滤器。同时选用了BKDHash、RSHash、APHash这三个哈希函数进行处理。

通过此次课程设计，我熟悉并掌握了布隆过滤器的数据结构，并能够用QT实现简单的GUI图形界面编程。在遇到问题时，学会了搜索官方文档去解决遇到的库调用问题。同时本次课程设计代码量较大，我分成了多个文件编写程序，不同文件实现不同的类，完成相应的功能。对于此课程设计项目，采用了CMake工具跨平台编译多个文件。

**（八）对课程设计的建议**

本人在此次课程设计中受益匪浅，通过此次课程设计，加深理解了数据结构的重要性，同时在解决问题的过程中，新学习到了上述技能。从被动学习知识的过程到自己主动发现问题并解决问题的进步，对于一个功能想法，去搜索是否有相应的库能够提供基础功能实现。对于先进知识，要有能够阅读英文文档的能力，更便于我们去解决问题。

从本人在此次课程设计的点滴成长可见，学校开设课程设计，有助于我们计算机专业学生进一步巩固所学，用所学去实现小功能、小项目，提升自我，并满足学生学习兴趣。

**（九）附录⎯⎯源程序**

此次课程设计项目本人已上传至我的github公有仓库中，可以更加方便进行获取查看，可以通过git获取我的整个项目源码，这里给出链接，打开链接即可：<https://github.com/ZonePG/NotePad>。

这里给出主要的9个文件(main.cpp、mainwindow.h、mainwindow.cpp、mainwindow.ui、myhighlight.h、myhighligh.cpp、bloomfilter.h、bloomfilter.cpp、CMakeLists.txt)源程序：

1. **main.cpp**

#include "mainwindow.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}

1. **mainwindow.h**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QFile>

#include <QFileDialog>

#include <QTextStream>

#include <QMessageBox>

#include <QtPrintSupport/QPrinter>

#include <QtPrintSupport/QPrintDialog>

#include <QtCore>

#include "myhighlight.h"

QT\_BEGIN\_NAMESPACE

namespace Ui { class MainWindow; }

QT\_END\_NAMESPACE

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

MainWindow(QWidget \*parent = nullptr);

~MainWindow();

private slots:

void on\_actionNew\_triggered();

void on\_actionOpen\_triggered();

void on\_actionSave\_as\_triggered();

void on\_actionExit\_triggered();

void on\_actionCopy\_triggered();

void on\_actionPaste\_triggered();

void on\_actionCut\_triggered();

void on\_actionUndo\_triggered();

void on\_actionRedo\_triggered();

void on\_actionC\_triggered();

void on\_actionJava\_triggered();

void on\_actionPython\_triggered();

private:

Ui::MainWindow \*ui;

QString currentFile = "";

// set highlight

MyHighlighter \*highlighter;

};

#endif // MAINWINDOW\_H

1. **mainwindow.cpp**

#include "mainwindow.h"

#include "./ui\_mainwindow.h"

#include <iostream>

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

this->setCentralWidget(ui->textEdit);

this->setWindowTitle("NotePad Powered by: ZONEPG");

ui->textEdit->setFont(QFont(tr("Consolas"), 11));

highlighter = new MyHighlighter(ui->textEdit->document());

// After this my highlightBlock() function will be called automatically whenever necessary.

}

MainWindow::~MainWindow()

{

delete ui;

}

void MainWindow::on\_actionNew\_triggered()

{

currentFile.clear();

ui->textEdit->setText(QString());

}

void MainWindow::on\_actionOpen\_triggered()

{

QString fileName = QFileDialog::getOpenFileName(this, "Open the file");

QFile file(fileName);

currentFile = fileName;

if (!file.open(QIODevice::ReadOnly | QFile::Text)) {

QMessageBox::warning(this, "Warning", "Cannot open file : " + file.errorString());

return;

}

setWindowTitle(fileName);

QTextStream in(&file);

QString text = in.readAll();

ui->textEdit->setText(text);

file.close();

}

void MainWindow::on\_actionSave\_as\_triggered()

{

QString fileName = QFileDialog::getSaveFileName(this, "Save as");

QFile file(fileName);

if (!file.open(QFile::WriteOnly | QFile::Text)) {

QMessageBox::warning(this, "Warning", "Cannot save file : " + file.errorString());

return;

}

currentFile = fileName;

setWindowTitle(fileName);

QTextStream out(&file);

QString text = ui->textEdit->toPlainText();

out << text;

file.close();

}

void MainWindow::on\_actionExit\_triggered()

{

QApplication::quit();

}

void MainWindow::on\_actionCopy\_triggered()

{

ui->textEdit->copy();

}

void MainWindow::on\_actionPaste\_triggered()

{

ui->textEdit->paste();

}

void MainWindow::on\_actionCut\_triggered()

{

ui->textEdit->cut();

}

void MainWindow::on\_actionUndo\_triggered()

{

ui->textEdit->undo();

}

void MainWindow::on\_actionRedo\_triggered()

{

ui->textEdit->redo();

}

void MainWindow::on\_actionC\_triggered()

{

highlighter->switchCpp();

}

void MainWindow::on\_actionJava\_triggered()

{

highlighter->switchJava();

}

void MainWindow::on\_actionPython\_triggered()

{

highlighter->switchPython();

}

1. **mainwindow.ui**

<?xml version="1.0" encoding="UTF-8"?>

<ui version="4.0">

<class>MainWindow</class>

<widget class="QMainWindow" name="MainWindow">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>555</width>

<height>409</height>

</rect>

</property>

<property name="windowTitle">

<string>MainWindow</string>

</property>

<widget class="QWidget" name="centralwidget">

<widget class="QWidget" name="verticalLayoutWidget">

<property name="geometry">

<rect>

<x>9</x>

<y>9</y>

<width>541</width>

<height>331</height>

</rect>

</property>

<layout class="QVBoxLayout" name="verticalLayout">

<item>

<widget class="QTextEdit" name="textEdit"/>

</item>

</layout>

</widget>

</widget>

<widget class="QMenuBar" name="menubar">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>555</width>

<height>31</height>

</rect>

</property>

<widget class="QMenu" name="menuFile">

<property name="title">

<string>File</string>

</property>

<addaction name="actionNew"/>

<addaction name="actionOpen"/>

<addaction name="actionSave\_as"/>

<addaction name="actionExit"/>

</widget>

<widget class="QMenu" name="menuEdit">

<property name="title">

<string>Edit</string>

</property>

<addaction name="actionCopy"/>

<addaction name="actionPaste"/>

<addaction name="actionCut"/>

<addaction name="actionUndo"/>

<addaction name="actionRedo"/>

</widget>

<widget class="QMenu" name="menuabout">

<property name="title">

<string>About</string>

</property>

<addaction name="actionauthor\_ZouPeng"/>

</widget>

<widget class="QMenu" name="menuHighLight\_Language">

<property name="title">

<string>Switch</string>

</property>

<addaction name="actionC"/>

<addaction name="actionJava"/>

<addaction name="actionPython"/>

</widget>

<addaction name="menuFile"/>

<addaction name="menuEdit"/>

<addaction name="menuHighLight\_Language"/>

<addaction name="menuabout"/>

</widget>

<widget class="QStatusBar" name="statusbar"/>

<action name="actionNew">

<property name="text">

<string>New</string>

</property>

</action>

<action name="actionOpen">

<property name="text">

<string>Open</string>

</property>

</action>

<action name="actionSave\_as">

<property name="text">

<string>Save as</string>

</property>

</action>

<action name="actionExit">

<property name="text">

<string>Exit</string>

</property>

</action>

<action name="actionCopy">

<property name="text">

<string>Copy</string>

</property>

</action>

<action name="actionPaste">

<property name="text">

<string>Paste</string>

</property>

</action>

<action name="actionCut">

<property name="text">

<string>Cut</string>

</property>

</action>

<action name="actionUndo">

<property name="text">

<string>Undo</string>

</property>

</action>

<action name="actionRedo">

<property name="text">

<string>Redo</string>

</property>

</action>

<action name="actionauthor\_ZouPeng">

<property name="text">

<string>Author: ZONEPG</string>

</property>

</action>

<action name="actionC">

<property name="text">

<string>C++</string>

</property>

</action>

<action name="actionJava">

<property name="text">

<string>Java</string>

</property>

</action>

<action name="actionPython">

<property name="text">

<string>Python</string>

</property>

</action>

</widget>

<resources/>

<connections/>

</ui>

1. **myhighlight.h**

#ifndef MYHIGHLIGHTER\_H

#define MYHIGHLIGHTER\_H

#include <QSyntaxHighlighter>

#include <QTextCharFormat>

#include <QRegularExpression>

#include <memory>

#include "bloomfilter.h"

class MyHighlighter : public QSyntaxHighlighter

{

public:

MyHighlighter(QTextDocument \*parent = nullptr);

~MyHighlighter();

// judge key whether exists in bloom filter or not.

bool isKeyExisted(QString key);

void switchCpp();

void switchJava();

void switchPython();

protected:

// highlight the word on TextEditor.

void highlightBlock(const QString &text) override;

// highliogh comment

void setCommentLine(const QString &text);

private:

QTextCharFormat myClassFormat;

QRegularExpression \*rule;

// my Bloom Filter member

BloomFilter \*BF;

};

#endif // MYHIGHLIGHTER\_H

1. **myhighlight.cpp**

#include "myhighlight.h"

using namespace std;

MyHighlighter::MyHighlighter(QTextDocument \*parent)

: QSyntaxHighlighter (parent)

{

// initial highlight rule

rule = new QRegularExpression("(\\b)(\\w+)(\\b)");

// initial highlight format

myClassFormat.setFontWeight(QFont::Bold);

myClassFormat.setForeground(Qt::blue);

BF = new BloomFilter();

}

MyHighlighter::~MyHighlighter()

{

}

bool MyHighlighter::isKeyExisted(QString key)

{

return BF->KeyExistInBitFilter((key.toStdString()));

}

void MyHighlighter::highlightBlock(const QString &text)

{

auto i = rule->globalMatch(text);

while (i.hasNext())

{

auto match = i.next();

auto begin = match.capturedStart();

auto end = match.capturedLength();

if (isKeyExisted(text.mid(begin, end)))

{

setFormat(begin, end, myClassFormat);

}

}

setCommentLine(text);

}

void MyHighlighter::setCommentLine(const QString &text)

{

QTextCharFormat multiLineCommentFormat;

multiLineCommentFormat.setForeground(Qt::red);

QRegularExpression startExpression("/\\\*");

QRegularExpression endExpression("\\\*/");

setCurrentBlockState(0);

int startIndex = 0;

if (previousBlockState() != 1)

startIndex = text.indexOf(startExpression);

while (startIndex >= 0) {

QRegularExpressionMatch endMatch;

int endIndex = text.indexOf(endExpression, startIndex, &endMatch);

int commentLength;

if (endIndex == -1) {

setCurrentBlockState(1);

commentLength = text.length() - startIndex;

} else {

commentLength = endIndex - startIndex

+ endMatch.capturedLength();

}

setFormat(startIndex, commentLength, multiLineCommentFormat);

startIndex = text.indexOf(startExpression,

startIndex + commentLength);

}

}

void MyHighlighter::switchCpp()

{

BF->import\_CPP\_KeyWords();

}

void MyHighlighter::switchJava()

{

BF->import\_Java\_keyWords();

}

void MyHighlighter::switchPython()

{

BF->import\_Python\_keyWords();

}

1. **bloomfilter.h**

#ifndef BLOOMFILTER\_H

#define BLOOMFILTER\_H

#include <string>

#include <vector>

const int MAXLEN = 875;

class BloomFilter {

private:

char bitFilter[MAXLEN];

std::vector<std::string> keyWords;

void initialBitFilter();

void refreshKeyWords();

void refreshBitFilter();

void importKeyToBloomFilter();

void readFileToArrays(const std::string str = "CPP.txt");

// Hash Func

int HashFunc(const std::string &str, int i);

int BKDRHash(const std::string &str);

int RSHash(const std::string &str);

int APHash(const std::string &str);

public:

BloomFilter();

~BloomFilter();

// user's control

void import\_CPP\_KeyWords();

void import\_Java\_keyWords();

void import\_Python\_keyWords();

bool KeyExistInBitFilter(const std::string &str);

};

#endif

1. **bloomfilter.cpp**

#include <iostream>

#include <fstream>

#include "bloomfilter.h"

BloomFilter::BloomFilter()

{

initialBitFilter();

import\_CPP\_KeyWords();

}

BloomFilter::~BloomFilter()

{

}

void BloomFilter::initialBitFilter()

{

for(int i=0; i<MAXLEN; i++)

{

bitFilter[i]=0;

}

}

void BloomFilter::refreshKeyWords()

{

keyWords.clear();

}

void BloomFilter::refreshBitFilter()

{

for (int i = 0; i < MAXLEN; i++)

bitFilter[i] = 0;

}

void BloomFilter::import\_CPP\_KeyWords()

{

refreshKeyWords();

refreshBitFilter();

readFileToArrays("CPP.txt");

importKeyToBloomFilter();

}

void BloomFilter::import\_Java\_keyWords()

{

refreshKeyWords();

refreshBitFilter();

readFileToArrays("Java.txt");

importKeyToBloomFilter();

}

void BloomFilter::import\_Python\_keyWords()

{

refreshKeyWords();

refreshBitFilter();

readFileToArrays("Python.txt");

importKeyToBloomFilter();

}

void BloomFilter::readFileToArrays(const std::string str)

{

std::ifstream in(str);

if (!in)

{

std::cout << "error opening the file." << std::endl;

}

std::string line;

while (in >> line)

keyWords.push\_back(line);

in.close();

}

void BloomFilter::importKeyToBloomFilter()

{

for (auto iter = keyWords.begin(); iter != keyWords.end(); iter++)

{

for (int i = 0; i < 3; i++)

{

auto bit = HashFunc((\*iter), i);

if(!((bitFilter[bit/8] >> bit%8) % 2))

{

bitFilter[bit/8] += 1 << bit % 8;

}

}

}

}

bool BloomFilter::KeyExistInBitFilter(const std::string &str)

{

for(int i=0; i<3; i++)

{

auto bit=HashFunc(str, i);

if(!((bitFilter[bit/8] >> bit%8) % 2))

return false;

}

return true;

}

int BloomFilter::HashFunc(const std::string &str, int i)

{

switch(i)

{

case 0:

return BKDRHash(str);

break;

case 1:

return RSHash(str);

break;

case 2:

return APHash(str);

break;

default:

break;

}

return 0;

}

int BloomFilter::BKDRHash(const std::string &str)

{

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

unsigned int hash = 0;

for (const auto &ch : str)

hash = hash \* seed + ch;

return (hash & 0x7FFFFFFF)%7000;;

}

int BloomFilter::RSHash(const std::string &str)

{

unsigned int hash = 0;

unsigned magic = 63689;

for (const auto &ch : str)

{

hash = hash \* magic + ch;

magic \*= 378551;

}

return hash % 7000;

}

int BloomFilter::APHash(const std::string &str)

{

unsigned int hash = 0;

int i = 0;

for (const auto &ch : str)

{

if ((i & 1) == 0)

hash ^= ((hash << 7) ^ (ch) ^ (hash >> 3));

else

hash ^= (~((hash << 11) ^ (ch) ^ (hash >> 5)));

i++;

}

return (hash & 0x7FFFFFFF)%7000;

}

1. **CMakeLists.txt**

cmake\_minimum\_required(VERSION 3.5)

project(NotePad LANGUAGES CXX)

set(CMAKE\_INCLUDE\_CURRENT\_DIR ON)

set(CMAKE\_AUTOUIC ON)

set(CMAKE\_AUTOMOC ON)

set(CMAKE\_AUTORCC ON)

set(CMAKE\_CXX\_STANDARD 11)

set(CMAKE\_CXX\_STANDARD\_REQUIRED ON)

SET(EXECUTABLE\_OUTPUT\_PATH ${PROJECT\_SOURCE\_DIR}/)

# QtCreator supports the following variables for Android, which are identical to qmake Android variables.

# Check http://doc.qt.io/qt-5/deployment-android.html for more information.

# They need to be set before the find\_package(Qt5 ...) call.

#if(ANDROID)

# set(ANDROID\_PACKAGE\_SOURCE\_DIR "${CMAKE\_CURRENT\_SOURCE\_DIR}/android")

# if (ANDROID\_ABI STREQUAL "armeabi-v7a")

# set(ANDROID\_EXTRA\_LIBS

# ${CMAKE\_CURRENT\_SOURCE\_DIR}/path/to/libcrypto.so

# ${CMAKE\_CURRENT\_SOURCE\_DIR}/path/to/libssl.so)

# endif()

#endif()

find\_package(Qt5 COMPONENTS Widgets REQUIRED)

if(ANDROID)

add\_library(NotePad SHARED

main.cpp

mainwindow.cpp

mainwindow.h

mainwindow.ui

myhighlight.h

myhighlight.cpp

bloomfilter.h

bloomfilter.cpp

)

else()

add\_executable(NotePad

main.cpp

mainwindow.cpp

mainwindow.h

mainwindow.ui

myhighlight.h

myhighlight.cpp

bloomfilter.h

bloomfilter.cpp

)

endif()

target\_link\_libraries(NotePad PRIVATE Qt5::Widgets)