关键字检索系统

#### 软件工程20级2班 2053300 胡锦晖

# 一、项目概述

## 1.1 项目背景

在如今互联网时代，通过浏览器或其他方式在线阅读一些文献、文章是经常会出现的场景，所以如何在海量的信息中快速检索到自己需要的内容也成为了一个十分有价值的问题。而本项目所模拟的关键字的检索方法是一个非常优秀的解决方案，通过提供关键字检索文章中包含关键字的部分能够帮助我们快速定位到我们所需要的信息。

## 1.2 项目目标

本项目模拟的是对一个文本文件进行关键字检索。建立一个文本文件，文件名和文本内容由用户输入；输入一个不含空格的关键字，统计并输出该关键字在文本中的出现次数。

# 二、实现思路

## 2.1 数据的存储结构

文本

中度可信度描述已自动生成本项目涉及的规模较大的数据为文本文件内容的存储，这里采用一个string类型变量存储。此外还需要一个string类型变量存储输入的文件名，以及用于读文件和写文件的文件流对象，具体如下图所示：

## 文本 描述已自动生成2.2 类的设计

## 2.3 功能实现

### 2.3.1 文件的读入

本项目的文件读入策略是屏幕上有字

描述已自动生成：逐个字符读取用户的输入，同步写入到文件（通过写入流对象fout）和存储文件内容的string类型变量text中（通过string的push\_back方法），直到遇到**结束符 ^**。

### 2.3.2 源文件的显示

由于无法直接将文件中的文本内容输出到终端，所以我们可以建立一个char型缓冲区数组buffer，现将文本内容按行输入到buffer中，再一行结束或缓冲区满的时候显示到终端，并清空缓冲区。缓冲区的大小通过一个int型常量定义：

重复以上操作直到文件结束（EOF）。

### 文本 描述已自动生成2.3.3 关键字的检索

这里需要说明的是，输入的待检索关键字中不能含有空格，因为空格会被认为是一次输入的结束而导致后半部分无法正常参与检索。此外，为了更好的模拟检索场景，通常对于一段文字所检索关键字的个数不止一个，所以程序不会仿照示例在一次检索之后退出，而是在一次检索后继续等待下一次检索的关键字，若想退出检索系统，可以输入**结束符 ^**。

## 2.4 核心算法

本项目的核心算法为克努特一莫里斯一普拉特算法，简称KMP算法。

模式匹配：子串在主串中的定位操作。

模式串：即要进行模式匹配的子串。

最大公共前后缀：字符串的前缀是指不包含最后一个字符的所有以第一个字符开头的连续子串；后缀是指不包含第一个字符的所有以最后一个字符结尾的连续子串；最大公共前后缀即为前缀与后缀交集的最长串。

### 2.4.1 基本思路

KMP模式匹配算法的基本思路就是通过将模式串向后移动比较指针（当前字符）之前的最大公共前后缀的长度，从而达到在每次移动模式串之后比较指针不回溯的效果。

### 2.4.2 预处理：next数组值的推导

由于一个模式串T各个位置（假设在该位置与主串出现了不匹配的情况）的最大公共前后缀长度不同且与比较的主串无关，则将这一长度的变化定义为一个数组next，该数组的长度即为串T的长度。求解方式用函数表示如下：

文本

描述已自动生成文本

描述已自动生成这里的第二种情况即为找到最大公共前后缀的长度，公共前后缀最短为，最长为，否则没有意义。

# 三、性能分析

## 3.1 相较于朴素模式匹配算法的优势

朴素模式匹配中在模式串移动一位之后，比较指针需要回到模式串最前端重新开始遍历，造成了大量重复匹配的情况，而KMP模式匹配避免了比较指针的移动，大大减少了重复遍历的情况。

## 3.2 复杂度分析

对于getNext函数来说，若subString的长度为，因只涉及到简单的单循环，其时间复杂度为，而由于值的不回溯，使得KMP函数中的效率得到了提高，while循环的时间复杂度为。因此，整个算法的时间复杂度为

通过时间复杂度的分析可以看出， KMP算法仅当模式串与主串之间存在许多“部分匹配“的情况下才体能现出优势，否则两者差异并不明显。

# 四、测试结果

## 4.1 Windows平台

操作系统：Windows11

编译器：g++ (GCC) 8.1.0

C++标准：c++17

## 4.2 Linux平台

操作系统：CentOS Linux release 8.5.2111

编译器：g++ (GCC) 8.5.0-4

C++标准：c++17

## 4.3 Mac平台

操作系统：macOS Monterey 12.0.1

编译器：clang++ 13.0.0

图形用户界面, 文本, 应用程序

描述已自动生成C++标准：c++17