项目说明文档

数据结构课程设计

——关键字检索系统

作 者 姓 名： 杨宇琨

学 号： 2252843

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

# 1 分析

## 1.1 背景分析

随着信息技术的发展，文本处理和信息检索成为日常工作中的重要任务。在处理大量文本数据时，需要有效地统计和分析关键字的出现情况，以便进行信息提取、关键词索引和文本挖掘等操作。因此，建立一个简单的文本文件关键字统计工具，可以帮助用户快速准确地获取文本中特定关键字的出现次数，提高工作效率。

## 1.2 功能分析

1. 用户可以通过键盘输入文件名，建立一个新的文本文件。这样做可以让用户自由选择需要处理的文本内容，增加了工具的灵活性和适用性。

2. 用户输入一个不含空格的关键字，程序将会对文本文件进行扫描，并统计输出该关键字在文本中的出现次数。这可以帮助用户快速了解关键字在文本中的分布情况，以及关键字的重要性。

3. 此工具的设计要求可以分成两个部分实现，即建立文本文件和关键字统计功能。这样的模块化设计可以使程序结构更加清晰，并便于后期的维护和扩展。

4. 用户友好的交互界面，包括清晰的输入提示、错误处理和统计结果的友好展示，以提高用户体验和工具的易用性。

5. 项目需要考虑异常情况的处理，例如用户输入错误的文件名、文件不存在或者无法读取等情况，需要给出相应的提示和处理方法，保证程序的稳定性和健壮性。

# 2 设计

## 2.1 数据结构设计

1. computeLPSArray函数中使用的数据结构：

- 数组：用于存储最长前缀和最长后缀匹配数组（LPS数组）lps。

2. KMPSearch函数中使用的数据结构：

- 数组：用于存储最长前缀和最长后缀匹配数组（LPS数组）lps，该数组在computeLPSArray函数中计算得到。

- 整型变量：用于记录关键字出现的次数count，以及控制循环的下标i和j。

3. countKeywordOccurrences函数中使用的数据结构：

- 字符数组：用于存储从文件中读取的每行文本内容line。

## 2.2 算法设计

1. computeLPSArray函数中的KMP算法：

- 通过比较pat[i]和pat[len]的值来更新LPS数组，其中i为当前处理的字符下标，len为当前已匹配的前缀和后缀长度。

- 时间复杂度为O(M)，其中M为关键字字符串的长度。

2. KMPSearch函数中的KMP算法：

- 使用已计算好的LPS数组，通过比较pat[j]和txt[i]的值来搜索关键字的出现情况。

- 时间复杂度为O(N+M)，其中N为文本字符串的长度，M为关键字字符串的长度。

## 2.3 系统设计

1. 输入输出：

- 系统通过标准输入获取用户输入的文件名和关键字，通过标准输出向用户展示关键字在文件中的出现次数。

2. 模块化设计：

- 系统将功能分解为计算LPS数组、KMP搜索和文件读取三个模块，使得系统结构清晰，易于维护和扩展。

3. 算法效率：

- 系统采用KMP算法实现关键字的快速搜索，能够有效提高搜索效率，尤其在大规模文本数据中效果更为明显。

4. 用户交互：

- 系统通过标准输入输出实现了与用户的简单交互，用户可以方便地输入文件名和关键字，同时获得搜索结果。

5. 异常处理：

- 系统在文件打开失败时给出相应的提示信息，增强了系统的鲁棒性和用户体验。

# 3 实现

## 3.1 KMP算法功能的实现

### 3.1.1 KMP算法功能流程

KMP算法逻辑流程如下：

1. 计算最长前缀和最长后缀匹配数组（LPS数组）：

- 首先需要计算出关键字的最长前缀和最长后缀匹配数组（LPS数组），可以通过computeLPSArray函数实现。

- LPS数组的长度与关键字的长度相同，lps[i]表示关键字的前缀子串[0, i]和后缀子串[0, i]的最大匹配长度。

2. 使用LPS数组进行关键字搜索：

- 通过KMP算法使用已经计算好的LPS数组在文本中搜索关键字的出现情况，可以通过KMPSearch函数实现。

- 设定两个指针i和j，分别表示文本和关键字的当前匹配位置。

- 遍历文本中的字符，逐个与关键字进行匹配，如果匹配成功则i和j分别向后移动，否则根据LPS数组更新j的位置。

3. 统计关键字出现的次数：

- 在搜索过程中，统计关键字在文本中出现的次数，即每当关键字完全匹配时，出现次数加一。

KMP算法通过预处理关键字构建LPS数组，在搜索时利用LPS数组进行匹配，避免了重复匹配的过程，提高了搜索的效率。

### 3.1.2 KMP算法功能核心代码

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

Function Name:computeLPSArray

Function:To compute Longest Proper Prefix which is also Suffix

Input Parameters:pattern string lenth of the string the match value

Return Value:void

Documentation：

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

void computeLPSArray(const char\* pat, int M, int\* lps)

{

int len = 0;

lps[0] = 0;

int i = 1;

while (i < M) {

if (pat[i] == pat[len]) {

len++;

lps[i] = len;

i++;

}

else {

if (len != 0)

len = lps[len - 1];

else {

lps[i] = 0;

i++;

}

}

}

}

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

Function Name:KMPSearch

Function:Using the KMP to search the string

Input Parameters:pattern string the txt file we input

Return Value:the times of appearance

Documentation：

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

int KMPSearch(const char\* pat, const char\* txt)

{

int M = strlen(pat);

int N = strlen(txt);

int lps[MAX\_SIZE];

computeLPSArray(pat, M, lps);

int count = 0;

int i = 0;

int j = 0;

while (i < N) {

if (pat[j] == txt[i]) {

j++;

i++;

}

if (j == M) {

count++;

j = lps[j - 1];

}

else if (i < N && pat[j] != txt[i]) {

if (j != 0)

j = lps[j - 1];

else

i = i + 1;

}

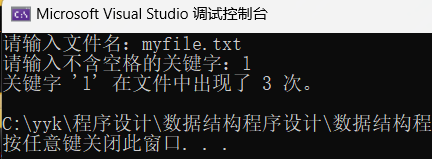
}

return count;

}

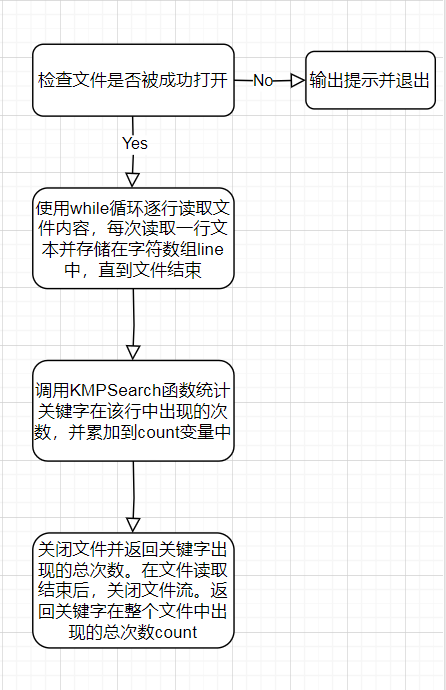
### 

### 3.1.3 KMP算法功能截屏示例



## 3.2 打开文件功能的实现

### 3.2.1 打开文件功能流程图



### 3.2.2 打开文件功能核心代码

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

Function Name:countKeywordOccurrences

Function:To open a file and calculate the time of appearance

Input Parameters:file name and the keyword

Return Value:the time of the appearance

Documentation：

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

int countKeywordOccurrences(const char\* filename, const char\* keyword)

{

std::ifstream file(filename);

char line[MAX\_SIZE];

int count = 0;

if (file.is\_open()) {

while (file.getline(line, MAX\_SIZE))

count += KMPSearch(keyword, line);

file.close();

}

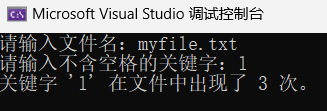
else

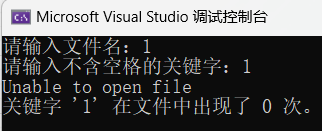
std::cout << "Unable to open file" << std::endl;

return count;

### }

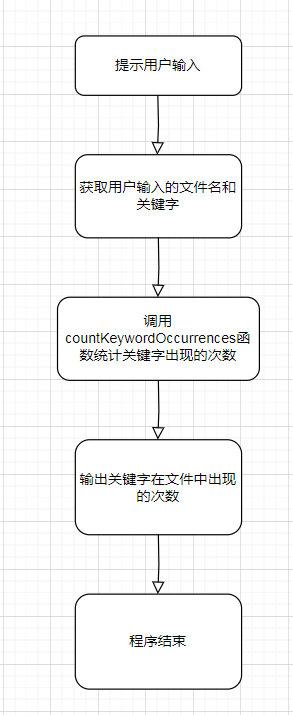
### 3.2.3 打开文件功能截屏示例





## 3.3 总体系统的实现

### 3.3.1 总体系统流程图



### 3.3.2 总体系统核心代码

int main() {

char filename[MAX\_SIZE];

char keyword[MAX\_SIZE];

std::cout << "请输入文件名：";

std::cin.getline(filename, MAX\_SIZE);

std::cout << "请输入不含空格的关键字：";

std::cin.getline(keyword, MAX\_SIZE);

int occurrences = countKeywordOccurrences(filename, keyword);

std::cout << "关键字 '" << keyword << "' 在文件中出现了 " << occurrences << " 次。" << std::endl;

return 0;

### }

### 3.3.3 总体系统截屏示例

