Project3 实验报告

22336044 陈圳煌

1、 程序功能说明

将文本信息存储在一个 file.txt 文档中，通过程序调用打开文本文件（文本文件中每行的词汇一律不跨行，每次读取一行则进行处理）。可以对文本进行同时查找多个字符串，并显示该字符串在文本中出现的行数（如果在一行中多次出现，仅显示一次），程序中提供了 KMP 算法和朴素匹配算法进行该操作，还可以返回文本的字符数（不包含空格）和显示各字母出现的频率的功能。

2、 程序运行及部分代码说明

首先建立一个 ifstream 类型的 ifs 变量，使用 ifs.open( )函数打开文件，并判断是否成功打开。

```
ifstream ifs;
ifs.open("file.txt");
if(!ifs.is_open()){
    cout << "文件打开失败！" << endl;
    return 0;
}
```

接着对文件进行操作，首先使用 eof（）函数判断文本中是否有内容。然后可以选择对文本的操作。

查找字符串：

（1） KMP 算法：通过 KMP 算法的原理，首先先求出 next 数组。

```
//kmp算法，求next数组
void getNext(string p,int *next) {
    int len=p.length();
    int j=0;
    next[0]=0;
    for (int i=1;i<len;i++){
        while(j>0 && p[i]!=p[j]){
            j=next[j-1];
        }
        if(p[i]==p[j]){
            j++;
        }
        next[i]=j;
    }
}
```

```
//通过kmp算法匹配字符串，如果在当行（s为目标串，为文件中的一行，
bool kmpSearch(string s, string p, bool isok, int& count) {
    int n=s.length();
    int m=p.length();
    int next[m];
    getNext(p, next);
    int j=0;
    for (int i=0;i<n;i++) {
        while(j>0 && s[i]!=p[j]){
            j=next[j-1];
        }
        if(s[i]==p[j]){
            j++;
        }
        if(j==m){
            count++;
            isok=1;
            j=next[j-1];
        }
    }
    return isok;
}
```

并根据 next 数组设计 KMP 模式匹配算法（上述代码段中 i 和 j 都是从 0 开始）

```
bool analysefile1(ifstream &file,string str,vector<int> &a,int &count){
    int len1=str.length();
    string line;
    int row=1;
    while(getline(file,line)){//利用getline进行以行为单位读取
        bool isok=0;//标记是否成功匹配到字符串
        if(kmpSearch(line,str,isok,count)){
            a.push_back(row);
        }
        row++;
    }
    if(count==0){
        return false;
    }else{
        return true;
    }
}
```

传入主串（s）和模式串（p），使用 getline（）函数逐行读取，通过 KMP 算法进行匹配，并用 count 记录模式串在主串中出现的次数，如果在当行出现则记录行数。

（2） 朴素匹配算法:

```
bool analysefile2(ifstream &file,string str,vector<int> &a,int &count){//
    int len1=str.length();
    string line;
    int row=1;
    while(getline(file,line)){//利用getline进行以行为单位读取
        bool isok=0;//标记是否成功匹配到字符串
        int len2=line.length();//为暴力朴素匹配
        for(int i=0;i<len2;i++){
            int k=i;
            for(int j=0;j<len1;j++){
                if(str[j]==line[k]){
                    k++;
                }else{
                    break;
                }
                if(j==len1-1){//当j遍历到最后一位就表示字符串匹配成功，is
                    isok=1;
                    count++;
                }
            }
        }
        if(isok==1){//当在当行匹配到字符串后，记录行数
            a.push_back(row);
        }
        row++;
    }
    if(count==0){
        return false;
    }else{
        return true;
    }
}
```

朴素匹配算法是将主串和模式串逐位比对，若有失配则主串和模式串一起返回上一位置再往后移，时间消耗上高于 KMP 算法。

求文件的总字符数:

```
int countfile(ifstream &file){//求文件的总字符数
    int sum=0;
    string line;
    while(getline(file,line)){//逐行读取
        int len=line.length();
        for(int i=0;i<len;i++){
            if(line[i]==' '){//去掉空格
                continue;
            }else{
                sum++;
            }
        }
    }
    return sum;
}
```

使用逐位判断记录字符数，遇到空格则跳过。

求文件中各字母出现的频率：

```
void frequency(ifstream &file){//求文件中各字母出现频率
    char al[26]={'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z'};//先预存26大小的数组
    int num[26];
    for(int i=0;i<26;i++){
        num[i]=0;
    }
    string line;
    while(getline(file,line)){
        int len=line.length();
        for(int i=0;i<len;i++){
            if(line[i]>='a' || line[i]<='z'){
                num[line[i]-'a']++;
            }else if(line[i]>='A' || line[i]<='Z'){
                num[line[i]-'A']++;//a和A都记录在表示a的数组里
            }
        }
    }
    for(int i=0;i<26;i++){
        cout << al[i] << " : " << num[i] << endl;
    }
}
```

首先先预存 26 位字母的数组以及记录出现次数的 num 数组，接着逐位查找，记录每个字母出现的次数。

3、 测试样例

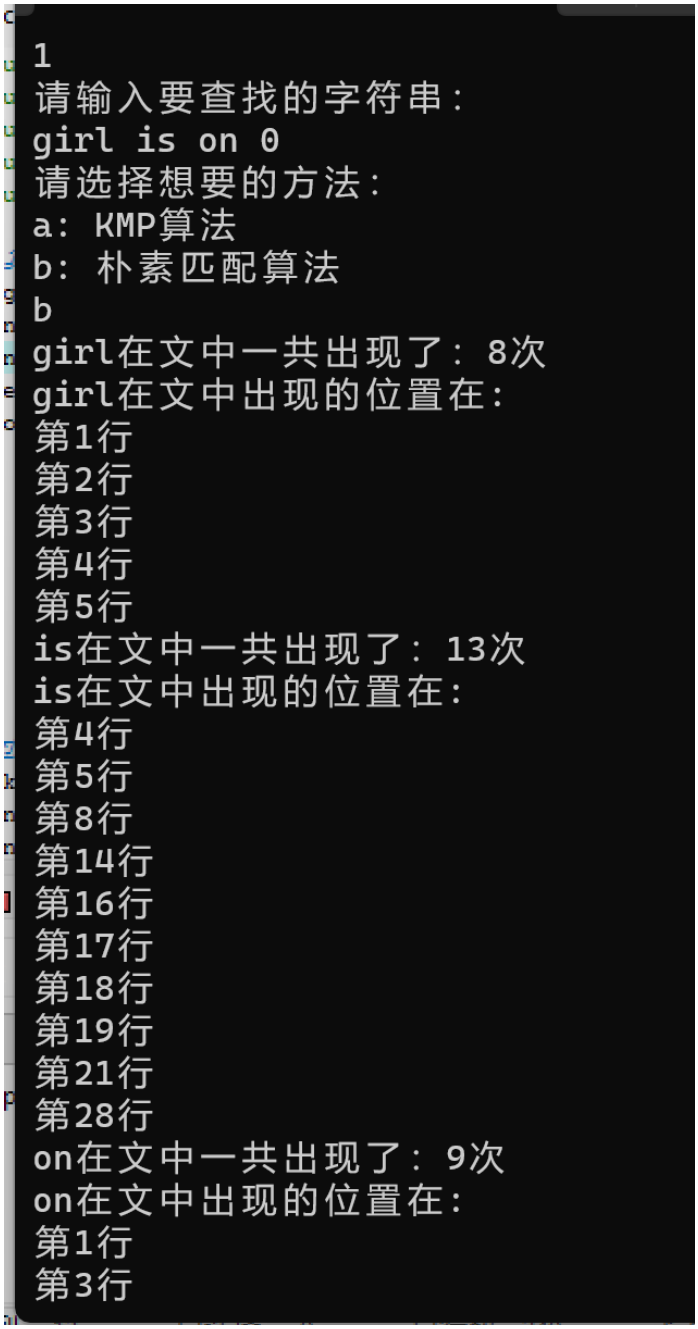
在 file.txt 文件中存入多段英文句子，使用程序打开该文件：

查找字符串：

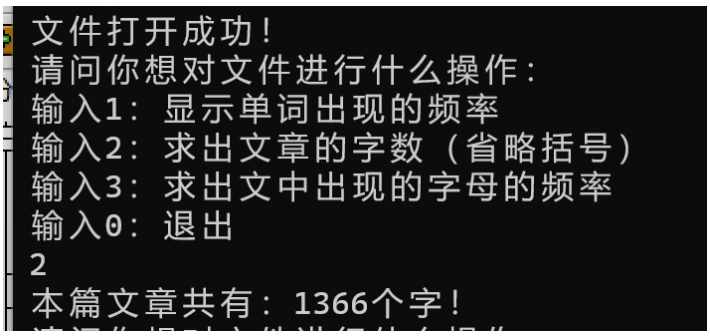
（1） KMP 算法：

```
输入0：退出
1
请输入要查找的字符串：
girl is on 0
请选择想要的方法：
a：KMP算法
b：朴素匹配算法
a
girl在文中一共出现了：8次
girl在文中出现的位置在：
第1行
第2行
第3行
第4行
第5行
is在文中一共出现了：13次
is在文中出现的位置在：
第4行
第5行
第8行
第14行
第16行
第17行
第18行
第19行
第21行
第28行
on在文中一共出现了：9次
on在文中出现的位置在：
```

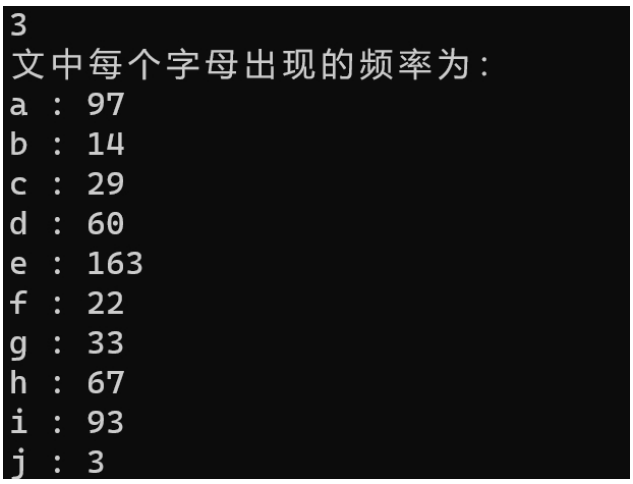
（2） 朴素匹配算法:

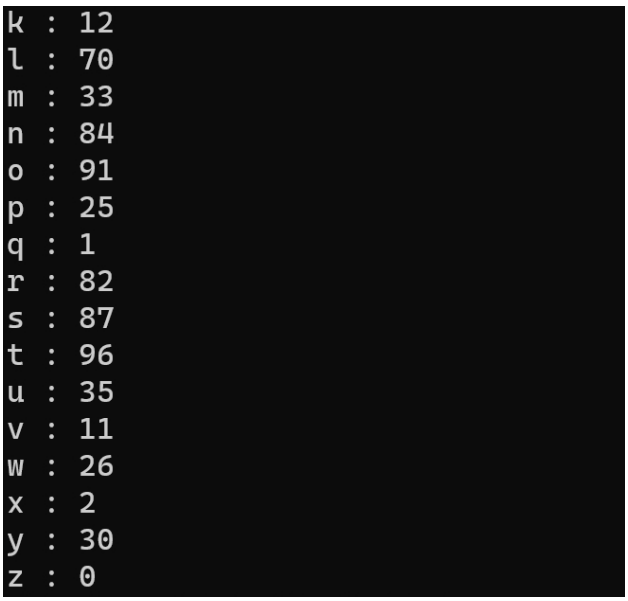


求文本中字符数:



求文本中各字母出现的频率:





经检验，程序输出正确。

4、 程序运行方式

首先在与可执行文件同一个文件夹中存入相应的 txt 文件，即可通过程序打开文本文件进行操作。当文件打开失败则会结束程序，文件打开成功后即可进行上述操作功能，并在执行操作前判断文件时候为空（或按行遍历过一遍后文件指针指到了最后）。在执行查找字符串后输入完成应输入一个“0”表示结束。

```
}else{
    cout << "请输入要查找的字符串: " << endl;
    queue<string> word;
    string w;
    cin >> w;
    while(w!="0"){
        word.push(w);
        cin >> w;
    }
```