数据结构上机报告

题目：文学研究助手

班级：04班 姓名： 学号： 完成日期：2020.11.17

一、需求分析

1.问题描述

文学研究人员需要统计某篇英文小说中某些形容词的出现次数和位置。一个实现这一目标的文字统计系统，称为“文学研究助手”。

1. 基本要求

英文小说存于一个文本文件中。待统计的词汇集合要一次输入完毕，即统计工作必须在程序的一次运行之后就全部完成。程序的输出结果是每个词的出现次数和出现位置所在的行的行号，格式自行设计。

3.测试数据

以本题目的C源程序模拟英文小说，C语言的保留字集作为待统计的词汇集。

4.程序功能

因程序实现目标为英文小说，故无需考虑对中文字词的匹配处理，对用户给定的每个单词，都要统计出出现次数和每次出现时单词的位置。

1. 概要设计:

按行读入文本与统计单词，将文本与所有单词分别存在两个二维数组中(文本按行存储、单词按个存储）。通过KMP算法实现单词匹配，将单词的位置数据存在数组中，并在匹配完成后按输入单词的顺序输出统计结果。

KMP算法与统计实现

1. 定义一个结构体类型存储单词的所在的行列信息。

typedef struct position { //结构体用于存放文中出现单词的位置信息

int row;

int num;

}pos;

2、主程序流程

a.读入同文件下txt文本形式的英文小说，然后将需要统计的单词数据读入到数组中。

b.读入过程中，进行单词的统计工作，并将统计结果保存在数组中。

c.按输入单词的顺序输出统计结果。

三、详细设计

匹配统计实现伪码

1、函数定义及功能

求next函数算法

void getnext(char a[],int next[],int n) { //KMP算法获取Next函数

int i = 0,j = -1;

next[0] = -1;

while(i < n) {

if(j == -1 || a[i] == a[j]) {

i++;

j++;

next[i] = j;

}

else

j = next[j];

}

}

按行匹配与统计单词位置函数（基于KMP算法）

int getnum(char a[],char b[],pos \*p,int \*t) {

//KMP算法实现 ，t用来指向单词出现次数的下标

int i = 0,j = 0,next[100] = {0},na,nb,sum = 0;

//sum统计单词本行出现的次数

nb = strlen(b);

getnext(b,next,nb);

na = strlen(a);

while(i < na) {

if(j == -1 || a[i] == b[j]) {

i++;

j++;

}

else

j = next[j];

if(j == nb) { //匹配一个单词后开始下一个单词的统计

p[\*t].num = i - j + 1; //记录单词列数

j = 0; //重置单词字符串，进行本行下一个单词的匹配

\*t = \*t + 1;

sum++;

}

}

return sum;

}

主程序处理部分：

1、读入数据与统计部分

while(!feof(fp)) {

fgets(a[i],sizeof(a[i]) - 1,fp); //按行输入文章

i++;

}

fclose(fp);

printf("请输入要查询的单词，每行一个，以#结束：\n");

gets(b[j]);

while(b[j][0] != '#') { //按行读入需要统计的单词

for(k = 0;k < i;k++) {

pq = sum[j];

sum[j] += getnum(a[k],b[j],p[j],t); //统计每行单词个数和位置

for(m = pq;m < sum[j];m++)

p[j][m].row = k + 1;

}

\*t = 0;

j++;

gets(b[j]);

}

for(i = 0;i < j;i++) { //按输入顺序输出各单词出现次数和位置

printf("\"");

printf("%s",b[i]);

printf("\" 单词在文中出现了 %d 次\n",sum[i]);

for(k = 0;k < sum[i];k++) {

printf("第 %d 个 \"",k+1);

printf("%s",b[i]);

printf("\" 的位置在第 %d 行第 %d 列",p[i][k].row,p[i][k].num);

}

}

四、调试分析

1、调试问题

a.初期实现程序时，用EOF判断读入结束，后发现与gets()的读入方式冲突，后改为#判定输入结束。

2、算法的时间复杂度分析和改进设想

算法的时间复杂度为O（M\*N），M为文章长度，N为单词数量，程序时间复杂度较高，而且适配性不强（不能处理中文字符），后期可以完善时间计算算法，只读取一遍文章，再采用utf-8编码读取文章，可以提高程序的可用性。

3、经验和体会

对KMP算法的使用有了更加深刻的了解，学习了基本的文本文件的操作方式。

五、用户使用说明

用户需将待求的txt文本文件放在源程序根目录中，运行所给文件夹下exe程序，并按次序输入待查询单词，即可得到输岀结果。

六、测试结果

输入样例

while

if

else

for

goto

break

#

输出样例

"while" 单词在文中出现了 4 次

第 1 个 "while" 的位置在第 12 行 第 2 列

第 2 个 "while" 的位置在第 28 行 第 2 列

第 3 个 "while" 的位置在第 51 行 第 2 列

第 4 个 "while" 的位置在第 58 行 第 2 列

"if" 单词在文中出现了 3 次

第 1 个 "if" 的位置在第 13 行 第 3 列

第 2 个 "if" 的位置在第 29 行 第 3 列

第 3 个 "if" 的位置在第 35 行 第 3 列

"else" 单词在文中出现了 2 次

第 1 个 "else" 的位置在第 18 行 第 3 列

第 2 个 "else" 的位置在第 33 行 第 3 列

"for" 单词在文中出现了 4 次

第 1 个 "for" 的位置在第 59 行 第 3 列

第 2 个 "for" 的位置在第 62 行 第 4 列

第 3 个 "for" 的位置在第 69 行 第 2 列

第 4 个 "for" 的位置在第 73 行 第 3 列

"goto" 单词在文中出现了 0 次

"break" 单词在文中出现了 0 次

1. 附录

#include<stdio.h>

#include<string.h>

typedef struct position { //结构体用于存放文中出现单词的位置信息

int row;

int num;

}pos;

void getnext(char a[],int next[],int n) { //KMP算法获取Next函数

int i = 0,j = -1;

next[0] = -1;

while(i < n) {

if(j == -1 || a[i] == a[j]) {

i++;

j++;

next[i] = j;

}

else

j = next[j];

}

}

int getnum(char a[],char b[],pos \*p,int \*t) { //KMP算法实现 ，t用来指向单词出现次数的下标

int i = 0,j = 0,next[100] = {0},na,nb,sum = 0; //sum统计单词本行出现的次数

nb = strlen(b);

getnext(b,next,nb);

na = strlen(a);

while(i < na) {

if(j == -1 || a[i] == b[j]) {

i++;

j++;

}

else

j = next[j];

if(j == nb) { //匹配一个单词后开始下一个单词的统计

p[\*t].num = i - j + 1; //记录单词列数

j = 0; //重置单词字符串，进行本行下一个单词的匹配

\*t = \*t + 1;

sum++;

}

}

return sum;

}

int main() {

FILE \*fp;

fp = fopen("text.txt","r");

pos p[100][1000];

char a[1000][300],b[100][100] = {0}; //a数组存储文章信息，b数组存储单词信息

int sum[1000] = {0},i = 0,j = 0,k,m,ps = 0,pq,\*t = &ps; //sum存储每个单词出现次数，i,j,k,m为循环变量

while(!feof(fp)) {

fgets(a[i],sizeof(a[i]) - 1,fp); //按行输入文章

i++;

}

fclose(fp);

printf("请输入要查询的单词，每行一个，以#结束：\n");

gets(b[j]);

while(b[j][0] != '#') { //按行读入需要统计的单词

for(k = 0;k < i;k++) {

pq = sum[j];

sum[j] += getnum(a[k],b[j],p[j],t); //统计每行单词个数和位置

for(m = pq;m < sum[j];m++)

p[j][m].row = k + 1;

}

\*t = 0;

j++;

gets(b[j]);

}

for(i = 0;i < j;i++) { //按输入顺序输出各单词出现次数和位置

printf("\"");

printf("%s",b[i]);

printf("\" 单词在文中出现了 %d 次\n",sum[i]);

for(k = 0;k < sum[i];k++) {

printf("第 %d 个 \"",k+1);

printf("%s",b[i]);

printf("\" 的位置在第 %d 行 第 %d 列\n",p[i][k].row,p[i][k].num);

}

}

return 0;

}