程序设计导论

课程设计实验报告

组长：吴坤尧（2016202144）

组员：黎春颖（2016202136）

范卓娅（2016202160）

张雅坤（2016202152）

钟家佳（2016202219）

目录

一、需求分析

- 程序应支持特定词典上的查询
  编写函数 void open_and_read（ ）;
- 程序应支持用户输入查询单词
- 程序应支持查询单词精确匹配
  编写函数 int exactsearch（ ）;
- 程序应支持查询单词模糊匹配
  编写函数 void fuzzysearch（ ）;

二、小组分工

黎春颖：（算法 1）读入文件
　　（算法 2）求向量函数 vector1,vector2;
范卓娅：（算法 1）预处理
　　（算法 2）相似度函数 similarity;
　　（算法 4）LCS 函数
张雅坤：（算法 1）快速排序 + 模糊输出
　　（算法 3）编辑距离函数 edit_distance
钟家佳：（算法 1）精确查找 + 模糊选择
　　（算法 3）优化编辑距离函数
吴坤尧：（算法 1）整体框架构建 + 主函数 + 求交并集函数

三、算法分析

3.1 算法 1（Jaccard 函数）

3.1.1 核心思想：交集、并集的比值表示相似度

共分为 5 部分

Part 1 读取文件

目标

1. 每个单词的长度 int word_len[2000]

2. 代替单词表的数组 char word[2000][20]

Part 2 处理单词

目标

单词表内每个单词对应的集合 char word_set[2000][26](0/1)

Part 3 精确查找

目标

1 输出该单词及对应序号

2 不存在则返回-1，进行模糊查找

Part 4 计算相似度 sim

目标

1 计算输入单词与单词表中每个单词的相似度 double sim[2000]

Part 5 输出结果

目标

按照相似度顺序、字典序输出结果

3.1.2 主要代码

```
/*****************************************************
     大作业：英文单词智能查询 jaccard 算法  2016.12.18
    黎春颖******读取文件函数，求单词表的单词长度
     范卓娅******预处理函数，标记单词表的单词字母出现
     钟家佳******精确查找函数
     吴坤尧******求交集函数，求并集函数，主函数
     张雅坤******精确度排序，模糊查找的输出
*****************************************************/
#include<stdio.h>
#include<string.h>
#include<stdlib.h>

int    MAX = 1991;
int    MAX_NUMBER = 5;
double     VALUE = 0.65;
int    word_len[2000] = {0};
double     sim[2000] = {0};
char        word[2000][20] = {'\0'};
char       word_set[2000][26] = {'\0'};
//*****************************************全局变量
void open_and_read ();
```

```
void preprocess ();
int exact_search (char input[]);
int intersection (char word_set1[], char input_set1[]);
int unio (char word_set2[], char input_set2[]);
void Print (double a[]);
int    partition (int index[], int left, int right);
void quicksort (int index[], int left, int right);
//****************************************函数声明
int main () {
	open_and_read ();//调用读取文件函数
	preprocess ();//调用预处理函数

	int i, j;
	char input[20] = {'\0'};
	char input_set[26] = {'0'};

	while (strcmp (input, "q") != 0) {
		for (i = 0; i < 20; i ++)	input[i] = '\0';
		for (i = 0; i < 26; i ++)	input_set[i] = '0';

		printf ("请输入要查询的单词（q to quit )：\n");
		scanf ("%s", &input);//获取单词
		if (strcmp (input, "q") == 0)	break;

		int len = strlen (input);//求单词长度
		for (i = 0; i < len; i ++) {//标记单词字母出现
			for (j = 0; j < 26; j ++) {
				if (input[i] == j + 'a' || input[i] == j + 'A')input_set[j]='1';
			}
		}

		int num = exact_search (input);//调用精确查找函数
		int g;
		if (num != -1)	printf ("您要查找的单词在第 %d 行。\n", num);
		else {
			printf ("没有找到该单词\n 模糊查询 ( 1 / 0 ) ？ : ");
			scanf ("%d", &g);
		}

		if (g == 1) {
			for (i = 0; i < MAX; i ++) {//求相似度
				int a	= intersection (word_set[i], input_set);//调用求交集函数
				int b	= unio (word_set[i], input_set);//调用求并集函数
				sim[i] = (double)a / b;
```

```
                    if (sim[i] == 1)  sim[i] = 0.999;
                }
                Print (sim);//调用相似度排序并输出函数
            }

            printf ("\n\n");
        }

        return 0;
}

void open_and_read () {//读取文件，求单词表单词长度
        FILE *in = fopen ("vocabulary.csv", "r");
        if (in == NULL) {
            printf ("cannot open the file!\n");
            exit (0);
        }

        int i, j;
        char str[20] = {'\0'};
        for (i = 0; i < MAX; i ++) {
            fgets (str, 20, in);
            word_len[i] = strlen (str) - 1;

            for (j = 0; str[j] != 10; j ++)
                word[i][j] = str[j];
            word[i][j] = '\0';
        }

        fclose (in);
}
void preprocess () {//预处理，标记单词表单词字母出现
        int i, j, k;
        for (i = 0; i < MAX; i ++) {
            for (j = 0; j < word_len[i]; j ++) {
                for (k = 0; k < 26; k ++) {
                    if (word[i][j] == k + 'a' || word[i][j] == k + 'A')
                        word_set[i][k] = '1';
                }
            }
        }
}

int exact_search (char input[]) {//精确查找 ,存在返回行数，不存在返回-1
```

```
	int i;
	for (i = 0; i < MAX; i ++) {
		if (strcmp (word[i], input) == 0)	return (i + 1);
	}
	return -1;
}

int intersection (char word_set1[], char input_set1[]) {//求交集
	int i;
	int counter = 0;
	for (i = 0; i < 26; i ++) {
		if (word_set1[i] == '1' && input_set1[i] == '1')	counter ++;
	}
	return counter;
}
int unio (char word_set2[], char input_set2[]) {//求并集
	int i;
	int counter = 0;
	for (i = 0; i < 26; i ++) {
		if (word_set2[i] == '1' || input_set2[i] == '1')counter ++;
	}
	return counter;
}

int	partition (double ary[], int index[], int left, int right) {//分区函数改编
	double k = ary[left];
	int	m = index[left];
	while (left < right) {
		while (	( (ary[right] == k && strcmp(word[index[right]], word[m]) >= 0) || ary[right] <
k ) && left < right	) {
			right --;
		}
		ary[left]	= ary[right];
		index[left] = index[right];

		while (	( (ary[left] == k && strcmp(word[index[left]], word[m]) <= 0) || ary[left] > k )
&& left < right	) {
			left ++;
		}
		ary[right]	= ary[left];
		index[right] = index[left];
	}
	ary[left]	= k;
	index[left] = m;
```

```
	return left;
}
void quicksort (double ary[], int index[], int left, int right) {//快排函数改编
	if (left >= right)	return;

	int p = partition (ary, index, left, right);
	quicksort (ary, index, left, p - 1);
	quicksort (ary, index, p + 1, right);
}
void Print (double a[]) {//相似度排序并输出函数
	int i, b[MAX];
	for (i = 0; i < MAX; i ++) //b[]代表单词数组的下标（位置）
		b[i] = i;

	quicksort (a, b, 0, MAX - 1);//调用快排函数

	int rank_width = 0, number = 0;
	for (i = 0; i < MAX_NUMBER; i ++) {//求列宽 rank_width
		if (word_len[b[i]] > rank_width)
			rank_width = word_len[b[i]];
		if (a[i] >= VALUE)
			number ++;
	}

	if (a[0] >= VALUE) {//判断输出
		printf ("最相近的前%d 个单词：\n", number);
		for (i = 0; i < number; i ++)
			printf ("%d. %-*s%.4lf\n", i + 1, rank_width + 1, word[b[i]], a[i]);
	}
	if (a[0] < VALUE) //判断不输出
		printf ("No Similar Word.\n");
}
```

3.2 算法 2（Cosine 函数）

3.2.1 核心思想：单词的字母向量表示

余弦值计算

part1.将算法 1 结果函数化

part2.求向量函数 vector1，vector2

part3.计算两向量夹角余弦

3.1.2 主要代码

```
/*************************************************************************
    大作业：英文单词智能查询 cosine 算法  2016.12.25
*************************************************************************/
#include <math.h>
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

const int  MAX = 1991;
double          VALUE = 0.65;
int          MAX_NUMBER = 5;

int    word_len[MAX] = {0};
char        word[MAX][20] = {'\0'};
char        input[30];
double    word_vector[MAX][26] = {0};//每个单词的特征向量
double    input_vector[26] = {0};
double    sim[MAX] = {0};
//*********************************************************全局变量
void open_and_read ();
void preprocess ();
void search ();
int    exactsearch ();
void fuzzysearch ();
void similarity    ();
char change    (char x);
void vector1 (int    k);
void vector2 (char voc[], double input_vector[]);
void Print ();
int    partition (int index[], int left, int right);
void quicksort (int index[], int left, int right);
//*********************************************************函数声明
int main () {
      open_and_read ();   //调用读取文件函数
      preprocess ();        //调用预处理函数

      int i;
      while (strcmp (input, "q") != 0) {
            for (i = 0; i < 20; i ++)      input[i] = '\0';
            for (i = 0; i < 26; i ++)      input_vector[i] = 0;
```

```
        printf ("请输入要查询的单词（q to quit )：\n");
        scanf ("%s", &input);    //获取单词
        if (strcmp (input, "q") == 0)    break;

        search ();        //调用单词查找函数
        printf ("\n\n\n");
    }

    return 0;
}

void open_and_read () {    //读取文件，求单词表单词长度
    FILE *in = fopen ("vocabulary.csv", "r");
    if (in == NULL) {
        printf ("cannot open the file!\n");
        exit (0);
    }

    int i, j;
    char str[20] = {'\0'};
    for (i = 0; i < MAX; i ++) {
        fgets (str, 20, in);
        word_len[i] = strlen (str) - 1;    //得到单词表单词的长度

        for (j = 0; str[j] != 10; j ++)
            word[i][j] = str[j];    //得到单词表单词的字符串

        word[i][j] = '\0';
    }

    fclose (in);
}
void preprocess () {//预处理
    int i;
    for (i = 0; i < MAX; i ++)
        vector1 (i);    //调用函数，求单词表每个单词的特征向量
}

void search () {//单词查找函数
    int i, a, g = 0;
    for(i = 0; i < strlen (input); i ++)
        g += isalpha (input[i]);    //判断是否全部非字母
```

```
    if (g != 0)
        a = exactsearch ();          //调用精确查找函数，得到返回值-1 / 0 / 1

    if (a == 1)
        fuzzysearch ();              //调用模糊查找函数
}

void vector1 (int k) {   //输入单词序数   在 preprocess 中使用   求所有单词的特征向量
    int   i, sum = 0;
    char transfered[20];
    for ( i = 0; i < word_len[k]; i ++)
        transfered[i] = change(word[k][i]);

    for ( i = 0; i < word_len[k]; i ++) {
        if (transfered[i]<'a'&&transfered[i]>'z') continue;

        int temp = transfered[i] - 'a';
        word_vector[k][temp] += transfered[i];

        if ( i != word_len[k] - 1) {
            double t = transfered[i] - transfered[i + 1];
            word_vector[k][temp] += fabs (t);
        }
    }

    for (i = 0; i < 26; i ++)   //归一化
        sum   += word_vector[k][i];
    for (i = 0; i < 26; i ++)
        word_vector[k][i] /= sum;
}
void vector2 (char voc[], double input_vector[]) {   //求所输入单词的特征向量 input_vector
    int i, temp = strlen (voc);
    double sum = 0;
    char     tra[20];

    for ( i = 0; i < temp; i ++)
        tra[i] = change (voc[i]);

    for ( i = 0; i < temp; i ++) {
        if (tra[i]<'a'|| tra[i]>'z') continue;

        int character = tra[i] - 'a';
        input_vector[character] += tra[i];
```

```
            double t =    tra[i] - tra[i + 1];
            if ( i != temp - 1)      input_vector[character] += fabs(t);
        }

        for (i = 0; i < 26; i ++) //归一化
            sum += input_vector[i];
        for (i = 0; i < 26; i ++)
            input_vector[i] /= sum;
}
char change (char x) {    //转换大小写
        if ( x >= 'A' && x <= 'Z')    x = x - 'A' + 'a';
        return x;
}

int exactsearch () {    //精确查找并输出函数
        int i;
        for (i = 0; i < MAX; i ++) {
            if (strcmp (word[i], input) == 0) {
                printf ("您要查找的单词在第  %d  行。\n", i + 1);
                return -1;
            }
        }
        printf ("没有找到该单词\n 模糊查询  ( 1 / 0 )  ？ : ");
        scanf ("%d", &i);
        return i;
}

void fuzzysearch () {    //模糊查找函数
        vector2 (input, input_vector);    //调用函数，求输入单词特征向量

        int i;
        for (i = 0; i < MAX; i ++)
            similarity ();    //调用求相似度函数

        Print ();    //调用函数，排序输出
}
void similarity () {    //相似度函数
        double sum[MAX] = {0}, wf[MAX] = {0}, ws = 0;
        int i, j;
        for (i = 0; i < MAX; i ++) {    //求单词表中每个单词的特征向量与输入单词特征向量的数量积 sum[]
            for (j = 0; j < 26; j ++)
                sum[i] += word_vector[i][j] * input_vector[j];
        }
```

```
	for (i = 0; i < MAX; i ++) {	//求单词表中向量模的平方 wf[]
		for (j = 0; j < 26; j ++)
			wf[i] += word_vector[i][j] * word_vector[i][j];
	}
	for (i = 0; i < 26; i ++)	//求输入向量模的平方 ws
		ws += input_vector[i] * input_vector[i];

	for (i = 0; i < MAX; i ++)	//求相似度 sim[]
		sim[i] = sum[i] / (sqrt (wf[i] * ws));
}
void Print () {	//相似度排序并输出函数
	int i, b[MAX];
	for (i = 0; i < MAX; i ++)	b[i] = i;//b[]代表单词数组的下标（位置）

	quicksort (b, 0, MAX - 1);//调用快排函数

	int rank_width = 0, number = 0;
	for (i = 0; i < MAX_NUMBER; i ++) {//求列宽 rank_width
		if (word_len[b[i]] > rank_width)
			rank_width = word_len[b[i]];
		if (sim[i] > VALUE)
			number ++;
	}

	if (sim[0] >= VALUE) {//判断输出
		printf ("最相近的前%d 个单词：\n", number);
		for (i = 0; i < number; i ++)
			printf ("%d. %-*s%.4lf\n", i + 1, rank_width + 1, word[b[i]], sim[i]);
	}
	if (sim[0] < VALUE)	//判断不输出
		printf ("No Similar Word.\n");
}
int	partition (int index[], int left, int right) {//分区函数改编
	double k = sim[left];
	int	m = index[left];
	while (left < right) {
		while (	( (sim[right] == k && strcmp(word[index[right]], word[m]) >= 0) || sim[right] <
k ) && left < right	)
			right --;

		sim[left]	= sim[right];
		index[left] = index[right];

		while (	( (sim[left] == k && strcmp(word[index[left]], word[m]) <= 0) || sim[left] > k )
```

```
&& left < right    )
                left ++;

            sim[right]    = sim[left];
            index[right] = index[left];
        }
        sim[left]    = k;
        index[left] = m;
        return left;
}
void quicksort (int index[], int left, int right) {    //快排函数改编
        if (left >= right)        return;

        int p = partition (index, left, right);
        quicksort (index, left, p - 1);
        quicksort (index, p + 1, right);
}
```

3.3 算法 3（Edit Distance 函数）

3.3.1 核心思想： 将算法 2 中 cosine 函数替换为编辑距离函数

3.3.2 主要代码

```
/*************************************************
    大作业：英文单词智能查询编辑距离算法       2016.12.25
*************************************************/
#include <math.h>
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

const int  MAX = 1991;
double          VALUE = 0.5;
int          MAX_NUMBER = 5;

int         input_len = 0;
int    word_len[MAX] = {0};
int         distance[MAX] = {0};
char        input[30];
char        word[MAX][20] = {'\0'};
```

```
double    sim[MAX] = {0};
//*************************************************************全局变量
void open_and_read ();
void search ();
int    exactsearch ();
void fuzzysearch ();
void similarity    ();
int    edit_distance (char *a, char *b, int i, int j);
int    min_of_three    (int x, int y, int z);
void Print ();
int    partition (int index[], int left, int right);
void quicksort (int index[], int left, int right);
//*************************************************************函数声明
int main () {
      open_and_read ();    //调用读取文件函数

      int i;
      while (strcmp (input, "q") != 0) {
            for (i = 0; i < 20; i ++)       input[i] = '\0';
            for (i = 0; i < MAX; i ++)   distance[i] = 0;

            printf ("请输入要查询的单词（q to quit )：\n");
            scanf ("%s", &input);//获取单词
            if (strcmp (input, "q") == 0)     break;

            input_len = strlen (input);
            search ();//调用单词查找函数
            printf ("\n\n\n");
      }

      return 0;
}

void open_and_read () {//读取文件，求单词表单词长度
      FILE *in = fopen ("vocabulary.csv", "r");
      if (in == NULL) {
            printf ("cannot open the file!\n");
            exit (0);
      }

      int i, j;
      char str[20] = {'\0'};
      for (i = 0; i < MAX; i ++) {
            fgets (str, 20, in);
```

```
            word_len[i] = strlen (str) - 1;//得到单词表单词的长度

            for (j = 0; str[j] != 10; j ++)
                word[i][j] = str[j];//得到单词表单词的字符串

            word[i][j] = '\0';
        }

        fclose (in);
}
void search () {//单词查找函数
        int a, i, g = 0;
        for (i = 0; i < strlen (input); i ++)
            g += isalpha (input[i]);

        if (g == 0) {
            for (i = 0; i < MAX; i ++)          sim[i] = 0;
        }

        if (g != 0)
            a = exactsearch ();       //调用精确查找函数，得到返回值-1（已找到）/ 0（未找到且不查）/ 1（模糊查询）

        if (a == 1)
            fuzzysearch ();           //调用模糊查找函数
}

int exactsearch () {//精确查找并输出函数
        int i;
        for (i = 0; i < MAX; i ++) {
            if (strcmp (word[i], input) == 0) {
                printf ("您要查找的单词在第 %d 行。\n", i + 1);
                return -1;
            }
        }
        printf ("没有找到该单词\n 模糊查询（1 / 0）？: ");
        scanf ("%d", &i);
        return i;
}

int edit_distance (char *a, char *b, int i, int j) {
        if (j == 0)         return i;
        if (i == 0)         return j;
        if (a[i - 1] == b[j - 1])       return edit_distance (a, b, i - 1, j - 1);
```

```
    else                    return min_of_three (edit_distance (a, b, i - 1, j) + 1,
                                                 edit_distance (a, b, i, j - 1) + 1,
                                                 edit_distance (a, b, i - 1, j - 1) + 1);
}
int min_of_three (int x, int y, int z) {
    int min = x;
    if (min > y)      min = y;
    if (min > z)      min = z;
    return min;
}
void fuzzysearch () {//模糊查找函数
    int i;
    for (i = 0; i < MAX; i ++) {
        distance[i] = edit_distance (input, word[i], input_len, word_len[i]);
    }

    similarity ();//调用求相似度函数
    Print ();//调用函数，排序输出
}
void similarity () {//相似度函数
    int i;
    int maxlen;
    for (i = 0; i < MAX; i ++) {
        maxlen = input_len > word_len[i] ? input_len : word_len[i];
        sim[i] = 1 - (double) distance[i] / maxlen;
    }
}
void Print () {//相似度排序并输出函数
    int i, b[MAX];
    for (i = 0; i < MAX; i ++)   b[i] = i;//b[]代表单词数组的下标（位置）

    quicksort (b, 0, MAX - 1);//调用快排函数

    int rank_width = 0, number = 0;
    for (i = 0; i < MAX_NUMBER; i ++) {//求列宽 rank_width
        if (word_len[b[i]] > rank_width)
            rank_width = word_len[b[i]];
        if (sim[i] > VALUE)
            number ++;
    }

    if (sim[0] >= VALUE) {//判断输出
        printf ("最相近的前%d 个单词：\n", number);
        for (i = 0; i < number; i ++)
```

```
                printf ("%d. %-*s%.4lf\n", i + 1, rank_width + 1, word[b[i]], sim[i]);
        }
        if (sim[0] < VALUE)    //判断不输出
                printf ("No Similar Word.\n");
}
int    partition (int index[], int left, int right) {//分区函数改编
        double k = sim[left];
        int        m = index[left];
        while (left < right) {
                while (    ( (sim[right] == k && strcmp(word[index[right]], word[m]) >= 0) || sim[right] <
k ) && left < right    )
                        right --;

                sim[left]      = sim[right];
                index[left] = index[right];

                while (    ( (sim[left] == k && strcmp(word[index[left]], word[m]) <= 0) || sim[left] > k )
&& left < right    )
                        left ++;

                sim[right]      = sim[left];
                index[right] = index[left];
        }
        sim[left]      = k;
        index[left] = m;
        return left;
}
void quicksort (int index[], int left, int right) {//快排函数改编
        if (left >= right)        return;

        int p = partition (index, left, right);
        quicksort (index, left, p - 1);
        quicksort (index, p + 1, right);
}
```

3.4 算法 4（LCS 算法）

3.4.1 核心想法：将两个字符串的比较结果保存在一个二维数组中，求出最长公共字串的长度

3.4.2 主要代码

```
/*********************************************************************
    大作业：英文单词智能查询 LCS 算法       2016.12.27
*********************************************************************/
#include <math.h>
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

const int  MAX = 1991;
double          VALUE = 0.65;
int          MAX_NUMBER = 5;

int        input_len = 0;
int   word_len[MAX] = {0};
int        LCS_len[MAX] = {0};
char       input[30];
char       word[MAX][20] = {'\0'};
double    sim[MAX] = {0};
//************************************************************全局变量
void open_and_read ();
void search ();
int   exactsearch ();
void fuzzysearch ();
void similarity   ();
int   lcs (char *a, char *b);
void Print ();
int   partition (int index[], int left, int right);
void quicksort (int index[], int left, int right);
//************************************************************函数声明
int main () {
    open_and_read ();   //调用读取文件函数

    int i;
    while (strcmp (input, "q") != 0) {
```

```
        for (i = 0; i < 20; i ++)       input[i] = '\0';
        for (i = 0; i < MAX; i ++)   LCS_len[i] = 0;

        printf ("请输入要查询的单词（q to quit )：\n");
        scanf ("%s", &input);//获取单词
        if (strcmp (input, "q") == 0)    break;

        input_len = strlen (input);
        search ();//调用单词查找函数
        printf ("\n\n\n");
    }

    return 0;
}

void open_and_read () {//读取文件，求单词表单词长度
    FILE *in = fopen ("vocabulary.csv", "r");
    if (in == NULL) {
        printf ("cannot open the file!\n");
        exit (0);
    }

    int i, j;
    char str[20] = {'\0'};
    for (i = 0; i < MAX; i ++) {
        fgets (str, 20, in);
        word_len[i] = strlen (str) - 1;//得到单词表单词的长度

        for (j = 0; str[j] != 10; j ++)
            word[i][j] = str[j];//得到单词表单词的字符串

        word[i][j] = '\0';
    }

    fclose (in);
}
void search () {//单词查找函数
    int i, a = 0, g = 0;
    for (i = 0; i < strlen (input); i ++)
        g += isalpha (input[i]);

    if (g != 0)
        a = exactsearch ();      //调用精确查找函数，得到返回值-1（已找到）/ 0（未找到且不查）/ 1（模糊查询）
```

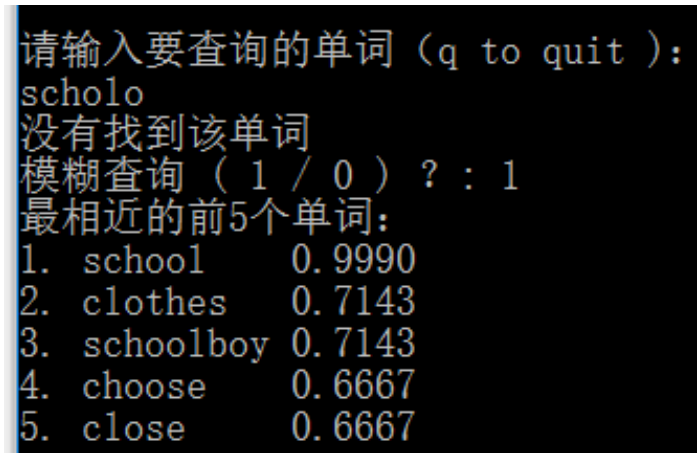
```
    if (a == 1)
        fuzzysearch ();          //调用模糊查找函数
}

int exactsearch () {//精确查找并输出函数
    int i;
    for (i = 0; i < MAX; i ++) {
        if (strcmp (word[i], input) == 0) {
            printf ("您要查找的单词在第 %d 行。\n", i + 1);
            return -1;
        }
    }
    printf ("没有找到该单词\n 模糊查询（1 / 0）？: ");
    scanf ("%d", &i);
    return i;
}

int lcs(char *a, char *b) {
    int i , j;
    int lena = strlen (a);
    int lenb = strlen (b);
    if (lena == 0 || lenb == 0)      return 0;

    int c[20][20];                  //c[i][j]记录 a[i]与 b[j]的 lcs 长度
    for (i = 1; i <= lena; i ++)        c[i][0] = 0;
    for (j = 1; j <= lenb; j ++)        c[0][j] = 0;
    c[0][0] = 0;
    for (i = 1; i <= lena; i ++) {
        for (j = 1; j <= lenb; j ++) {
            if (a[i-1] == b[j-1])
                c[i][j] = c[i-1][j-1] + 1;     //自底向上递归推算

            else
                c[i][j] = c[i-1][j] > c[i][j-1] ? c[i-1][j] : c[i][j-1];

        }
    }
    return c[lena][lenb];
}
void fuzzysearch () {//模糊查找函数
    int i;
    for (i = 0; i < MAX; i ++){
        LCS_len[i] = lcs (word[i], input);
```

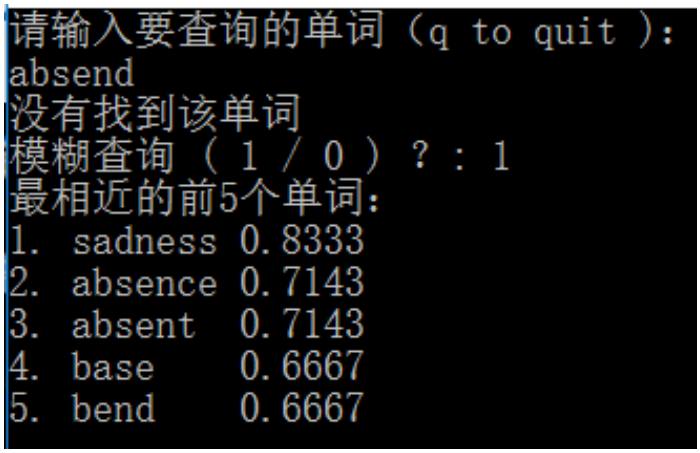
```
	}
	similarity ();//调用求相似度函数
	Print ();//调用函数，排序输出
}
void similarity () {//相似度函数
	int i;
	for (i = 0; i < MAX; i ++) {
		sim[i] = (double)2 * LCS_len[i] / (input_len + word_len[i]);
	}
}
void Print () {//相似度排序并输出函数
	int i, b[MAX];
	for (i = 0; i < MAX; i ++)  b[i] = i;//b[]代表单词数组的下标（位置）

	quicksort (b, 0, MAX - 1);//调用快排函数

	int rank_width = 0, number = 0;
	for (i = 0; i < MAX_NUMBER; i ++) {//求列宽 rank_width
		if (word_len[b[i]] > rank_width)
			rank_width = word_len[b[i]];
		if (sim[i] > VALUE)
			number ++;
	}

	if (sim[0] >= VALUE) {//判断输出
		printf ("最相近的前%d 个单词：\n", number);
		for (i = 0; i < number; i ++)
			printf ("%d. %-*s%.4lf\n", i + 1, rank_width + 1, word[b[i]], sim[i]);
	}
	if (sim[0] < VALUE)   //判断不输出
		printf ("No Similar Word.\n");
}
int   partition (int index[], int left, int right) {//分区函数改编
	double k = sim[left];
	int       m = index[left];
	while (left < right) {
		while (   ( (sim[right] == k && strcmp(word[index[right]], word[m]) >= 0) || sim[right] <
k ) && left < right   )
			right --;

		sim[left]     = sim[right];
		index[left] = index[right];

		while (   ( (sim[left] == k && strcmp(word[index[left]], word[m]) <= 0) || sim[left] > k )
```

```
&& left < right    )
                left ++;

            sim[right]      = sim[left];
            index[right] = index[left];
        }
        sim[left]      = k;
        index[left] = m;
        return left;
}
void quicksort (int index[], int left, int right) {//快排函数改编
        if (left >= right)        return;

        int p = partition (index, left, right);
        quicksort (index, left, p - 1);
        quicksort (index, p + 1, right);
}
```

四、结果分析

4.1 算法 1（Jaccard 函数）

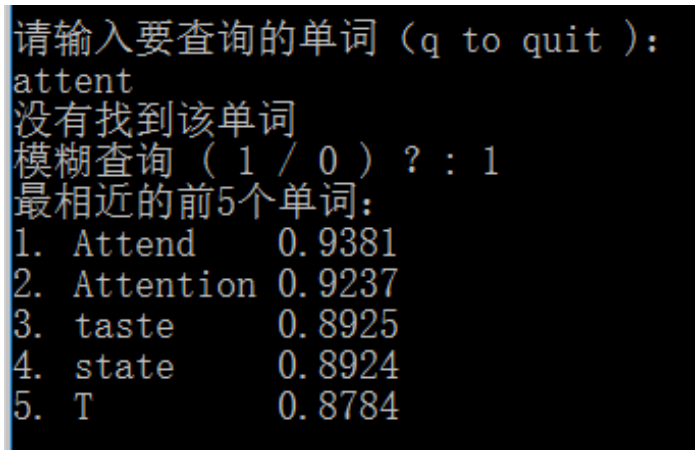


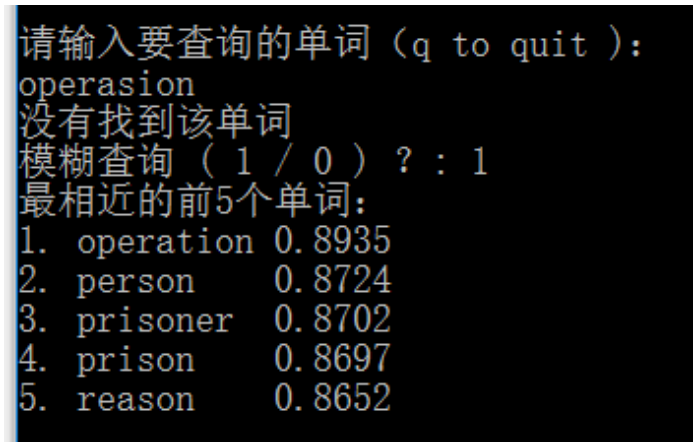


算法优点：思路简单、易实现

算法缺点：没有考虑到字母的顺序，没有考虑到字母重复出现，不符合“拼写错误”的一般规律。

4.2 算法 2（Cosine 函数）

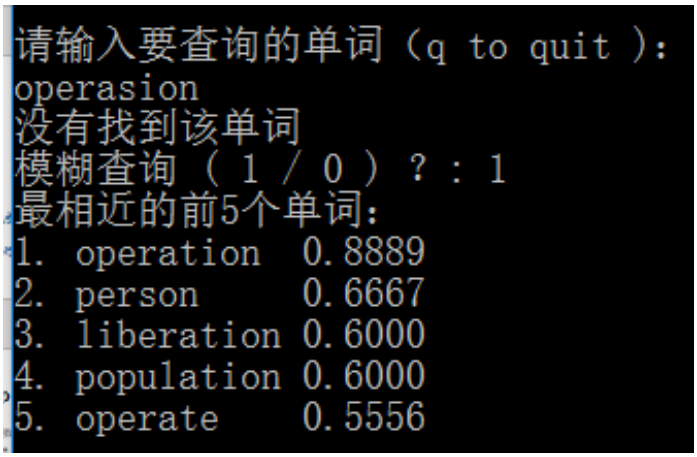


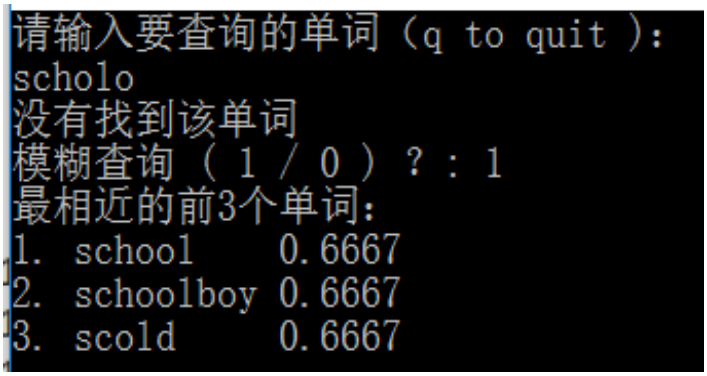


算法优点：考虑了字母顺序以及重复出现

算法缺点：计算量大，运行速度较长

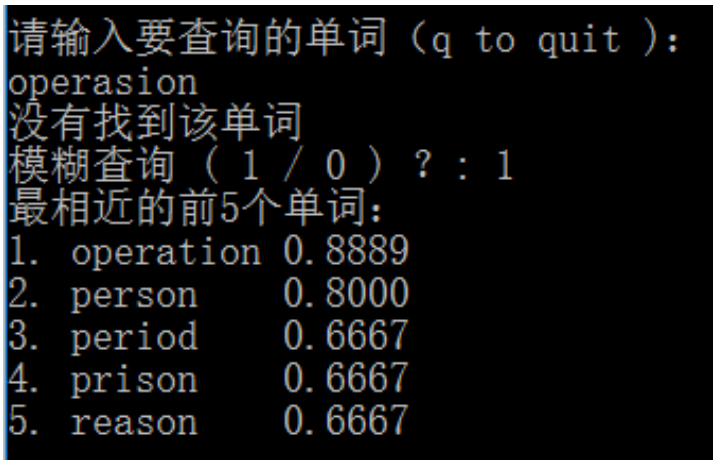
4.3 算法 3（Edit Distance 函数）

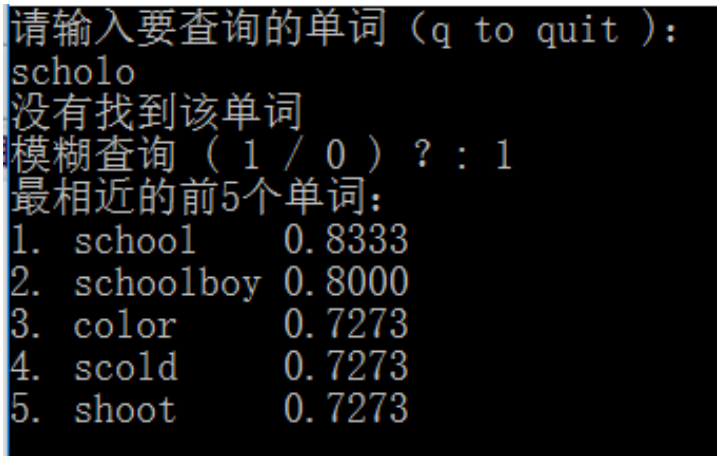




算法优点：符合“拼写错误”的一般规律，匹配程度高。
算法缺点：运行速度较长

4.4 算法 4（LCS 算法）





算法优点：符合“拼写错误”的一般规律，匹配程度高。
算法缺点：运行速度较长

五、问题与解决

函数：快速排序并输出函数 Print；quicksort；partition

问题：1.若只对相似度 sim[ ]进行排序，无法确定每个相似度对应的单词，不能对应输出；
2.优先排序相似度，在相似度相同的前提下进行字典排序；
3.在 printf 中使用%-*s 人为指定输出长度；

解决：1.增加一个下标数组 b[ ]，表示单词的序号，在快速排序相似度过程中，对下标数组进行同样的操作，最后输出下标对应的单词 word[b[ ]]；
2.在分区函数中重设分区标准,将值与枢纽点相同但对应单词字典序小的 sim 和值比枢纽点大的 sim 放在枢纽点前，将值与枢纽点相同但对应单词字典序大的 sim 和值比枢纽点小的 sim 放在枢纽点后；
3.输出格式左对齐并控制列宽为最大单词长度加一；

优点：1.解决对应输出问题的同时，不需改变原单词的顺序，耗时短；
2.将相似度快排与字典排序融合起来，代码简洁而高效；

六、修正与优化

张雅坤

读入文件函数 open_and_read

问题：用 gets（str）收单词时收入换行符\n，导致输出的单词与相似度不在同一行

解决：在将 str 赋给 word[ ][ ]时，控制不收\n，并置为'\0'

求相似度 cos 函数 similarity

问题：反复计算了输入单词向量的模，结果偏小

解决：改正函数

求单词向量函数 vector1,vector2
问题：对向量进行归一化处理时，反复归一化，导致结果偏小
解决：改正函数

钟家佳
主函数 main
问题：用 gets（input）收字符串被吞
解决：改用 scanf

吴坤尧
主函数 main
问题：多次查询时接收输入单词的字符串出现错误
解决：每次查询完进行初始化为\0

范卓娅
查询函数 search
问题：若输入字符均非字母，则查询费时费力且无意义
解决：增加变量 g 判断输入字符是否含字母
优点：省时省力