

程序设计导论

课程设计实验报告

组长：林子祺 (2016202234)

组员：宋晓林 (2016202149)

杨国兴 (2016202145)

杨敬栋 (2016202139)

叶挺茂 (2016202142)

目录

一、需求分析.....	2
二、小组分工.....	3
三、算法分析.....	4
3.1 算法 1 (Cosine 函数)	4
3.2 算法 2 (Edit Distance 函数)	5
四、结果分析.....	8
五、优化.....	10
六、感想体会.....	16
附录 1 源代码	18
附录 2 参考资料	26

一、需求分析

电子词典是日常生活中十分常用的工具。用户遇到不认识的单词时，只要打开电子词典，输入想要查询的单词便可得到想要的结果。随着科技的发展，网络搜索的盛行，各大搜索引擎也出现了搜索单词的功能，在其基础上，进一步出现了“模糊查询”的功能，即用户使用电子词典或搜索引擎时，如果出现拼写错误，程序会推测用户的真实意图（如下图）。



该方法依据用户所输入的单词，判断在字典中是否存在该单词，若不存在，则计算其相似度，输出相似度较高的单词。今天我们考虑使用 C 语言程序实现以下功能：

- 1、程序应支持用户在特定词典上的查询；
- 2、程序应支持用户输入查询单词；
- 3、程序应支持查询单词精确匹配；
- 4、程序应支持查询单词模糊匹配；
- 5、程序应支持多次查找；
- 6、程序应支持输出结果单词在特定词典中的所在行；
- 7、程序实现将查询历史写入指定文件。

二、小组分工

1、林子祺（2016202234）

主要负责：整体优化、用特征向量方法计算相似度；

2、宋晓林（2016202149）

主要负责：文件读入、日志输出；

3、杨国兴（2016202145）

主要负责：编辑距离函数；

4、杨敬栋（2016202139）

主要负责：输出部分；

5、叶挺茂（2016202142）

主要负责：精确匹配。

三、算法分析

3.1 算法 1 (Cosine 函数)

3.1.1 核心思想

这种方法实际作用是计算字母频率，即两个单词中频率相近的字母比例越高会被认为越相似，这种计算方法对于字母数较少的单词间的比较更为精准，但碰到用户输入单词符合以下条件：1. 虽然单词较长但是相异字母数较少 2. 各字母的频率接近，就会出现一个含有相同字母的短词被认为与该长词相似。

3.1.2 主要代码

```
double compute_cos(double *ary1, double *ary2, int n)
{
    double quasum1 = 0.0, quasum2 = 0.0, quasumOfDif = 0.0,
temp;

    double dis1, dis2, dis, cos;

    int i;

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

        quasum1 += (ary1[i] * ary1[i]);

        quasum2 += (ary2[i] * ary2[i]);

        temp = fabs(ary1[i] - ary2[i]);
```

```

        quassumOfDif += (temp * temp);
    }

    dis1 = sqrt(quassum1);
    dis2 = sqrt(quassum2);
    dis = sqrt(quassumOfDif);
    cos = (dis1*dis1 + dis2*dis2 - dis*dis)/(2 * dis1 * dis2);
    return cos;
}

```

3.2 算法 1 (Edit Distance 函数)

3.2.1 核心思想

这种方法从编辑距离（一个整数）到相似度的实现方法是取编辑距离和输入词和对比词长度的平均值的比值，再用 1 减去这个比值，取平均值的原因是如果输入词较长而对比词是输入词当中的一部分时（当然不是开头的一部分，因为如果是开头的话可能是用户输入太多，这种情况会在后面讲），编辑距离值并不大而这两个词从常识判断应该是不相似的，这种情况下取平均值会使分母变小从而降低系统判定的相似度。在输入单词较长时精确度更高，在查询短词时的应用效果不理想，例如想要查询 red 却打成 rde，根据上述算法得到的编辑距离相似度只有 0.33，显然不合理。

3.1.2 主要代码

```

double editDistance(char *p, char *q) {

    int i, j;

    int len1 = strlen(p); //统计输入的单词长度
    int len2 = strlen(q); //统计对比单词的长度
    int step[len1+1][len2+1]; // 建立一个矩阵
    step[0][0] = 0;

    for(i=1; i <= len1; i++)

    {

        step[i][0] = i;
    }

    for(j=1; j <= len2; j++)

    {

        step[0][j] = j;
    }

    //限定边界的值

    int k, l;

    for(k=1; k <= len1; k++)

    {

        for(l=1; l <= len2; l++)

        {

            if(*(p+k-1) == *(q+l-1))

            {

```

```

        step[k][1]=step[k-1][1-1];
    }
    else
    {
        int min;
        min = min_of_three(step[k-1][1-1], step[k-
1][1], step[k][1-1]);
        step[k][1] = min+1;
    }
}
}

//逐个填矩阵

double edit_dis =
(double)step[len1][len2]/(double)(len1/2 + len2/2); //算编辑
距离

double result = 1-edit_dis;

return result;
}

```


四、结果分析

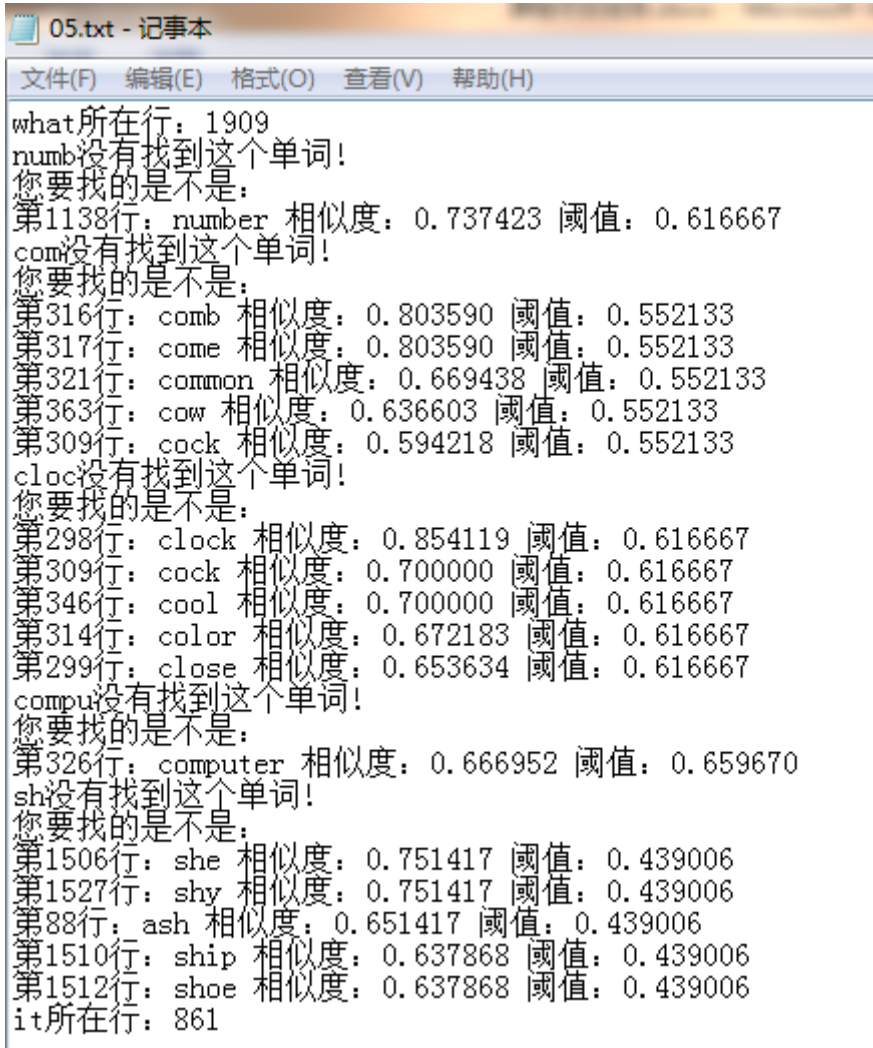
1、精确匹配

```
请您输入您想打开的文件路径（注意文件类型）：  
vocabulary.csv  
请您输入想要将输出保存的文件：  
search_history.txt  
请输入您想要搜索的单词（不区分大小写）（PS：输入两个0退出程序）：  
computer  
所在行： 328  
science  
所在行： 1466
```

2、模糊匹配

```
compu  
没有找到这个单词！  
您要找的是不是：  
第328行： computer 相似度： 0.666952 阈值： 0.659670  
histry  
没有找到这个单词！  
您要找的是不是：  
第785行： history 相似度： 0.883982 阈值： 0.690932  
artisst  
没有找到这个单词！  
您要找的是不是：  
第87行： artist 相似度： 0.892885 阈值： 0.714967  
data  
没有找到这个单词！  
您要找的是不是：  
第385行： date 相似度： 0.804923 阈值： 0.616667  
第395行： death 相似度： 0.653634 阈值： 0.616667
```

3、历史数据



```
05.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
what所在行: 1909
numb没有找到这个单词!
您要找的是不是:
第1138行: number 相似度: 0.737423 阈值: 0.616667
com没有找到这个单词!
您要找的是不是:
第316行: comb 相似度: 0.803590 阈值: 0.552133
第317行: come 相似度: 0.803590 阈值: 0.552133
第321行: common 相似度: 0.669438 阈值: 0.552133
第363行: cow 相似度: 0.636603 阈值: 0.552133
第309行: cock 相似度: 0.594218 阈值: 0.552133
clac没有找到这个单词!
您要找的是不是:
第298行: clock 相似度: 0.854119 阈值: 0.616667
第309行: cock 相似度: 0.700000 阈值: 0.616667
第346行: cool 相似度: 0.700000 阈值: 0.616667
第314行: color 相似度: 0.672183 阈值: 0.616667
第299行: close 相似度: 0.653634 阈值: 0.616667
compu没有找到这个单词!
您要找的是不是:
第326行: computer 相似度: 0.666952 阈值: 0.659670
sh没有找到这个单词!
您要找的是不是:
第1506行: she 相似度: 0.751417 阈值: 0.439006
第1527行: shy 相似度: 0.751417 阈值: 0.439006
第88行: ash 相似度: 0.651417 阈值: 0.439006
第1510行: ship 相似度: 0.637868 阈值: 0.439006
第1512行: shoe 相似度: 0.637868 阈值: 0.439006
it所在行: 861
```

五、优化

一、精确匹配部分；

我们通过二分查找在所给词典文件中检索单词，通过字符串的对比函数（strcmp）来实现精确查找功能；

二、在模糊匹配部分：

我们运用两种相似度计算方法。一是计算特征向量的夹角余弦值，二是计算编辑距离。前者的实际作用是计算字母频率，即两个单词中频率相近的字母比例越高会被认为越相似，这种计算方法对于字母数较少的单词间的比较更为精准，但碰到用户输入单词符合以下条件：

1. 虽然单词较长但是相异字母数较少 2. 各字母的频率接近，就会出现一个含有相同字母的短词被认为与该长词相似。后者从编辑距离（一个整数）到相似度的实现方法是取编辑距离和输入词和对比词长度的平均值的比值，再用 1 减去这个比值，取平均值的原因是如果输入词较长而对比词是输入词当中的一部分时（当然不是开头的一部分，因为如果是开头的话可能是用户输入太多，这种情况会在后面讲），编辑距离值并不大而这两个词从常识判断应该是不相似的，这种情况下取平均值会使分母变小从而降低系统判定的相似度。在输入单词较长时精确度更高，在查询短词时的应用效果不理想，例如想要查询 red 却打成 rde，根据上述算法得到的编辑距离相似度只有 0.33，显然不合理。

两种计算方法一种适用于长词，一种适用于短词，最后解决方法是根据输入单词取加权平均值，权值是根据输入单词的长度 n 的函数。目前在 \cos 值前选择的权值是 $0.9/\sqrt{n}$ ，而在编辑距离相似度前的权值是 $0.9-0.9/\sqrt{n}$ ，二者之和是 0.9 而非 1 ，这个在后面会解释。 \cos 前的权值随 n 增加而递减，后者的权值反之，这样就解决无法同时适用于两种情况的问题。

下面要说明上面亟待解决的两个问题，用户在查询某长单词的时候可能只记得该单词前面的一部分，而查询某短词的时候会以有后缀的长词作为寻找依据（比如不知道 *administrate* 而选择输入 *adminsitation*，即只知较长的名词形式或者形容词形式而不确定动词形式）。对于这两种需求，目前选择的解决办法是若输入词和对比词首字母相同则在最后的相似度值上 $+0.1$ ，这也解释了为什么两个权值之和为 0.9 。

关于阈值，想进一步补充说明。对于一个输入词以及一个对比词，人可以根据自己对拼音语言的认识说出它们相似、勉强相似或是根本不相及。阈值就是区分前两类和第三类的界线，阈值取决于输入单词的长度。阈值的计算方式是运用两种计算相似度的方法取临界情况下的值。目前的规定是对于一长度为 n 的单词（假定它有 n 个不同的字母构成，在这种情况下相等数量的字频差对相似度的影响最大），我们认为临界情况是 $1. \sqrt{n}/2$ 个字母在输入词和对比词中是只

存在于其一的 2. 二者的编辑距离是 $1.2 * \sqrt{n}$ （这是用一些样本测试的结果，不精确） 阈值取这两种情况下算得的相似度的平均值，低于阈值则被认为是在人的观念中不相似的。

在程序中我们使用了链表排序。所有相似度高于阈值的对比词都会被接入链表，链表排序是先按相似度从大到小，相似度相等则按字典序。选择链表排序的原因是超过阈值的对比词数量无法确定。输出链表时的规定如下：首先输出前五个，从第六个开始若其相似度与第五个相等则继续输出。原因是相似度相等的词在用户的需求中优先级应该是相等的，为了解决实际需求显然有必要输出。

（1）以下截图将说明首字母相似对搜索结果的影响：

图 1（未引入首字母相似机制）

图 2（引入首字母相似机制）

（2）以下截图将说明阈值机制对搜索结果的影响：

图 3（未引入阈值机制）

图 4（引入阈值机制）

```
C:\Users\Administrator\Documents\源代码\程设大作业\程设大作业.e...
请输入您想要搜索的单词（不区分大小写）（PS：按俩0退出程序）：
attent
没有找到这个单词！
您要找的是不是：
第100行：Attend 相似度：0.868126 阈值：0.690932
第1217行：patient 相似度：0.748055 阈值：0.690932
第101行：Attention 相似度：0.709724 阈值：0.690932
compu
没有找到这个单词！
numb
没有找到这个单词！
您要找的是不是：
第1144行：number 相似度：0.708248 阈值：0.616667
sh
没有找到这个单词！
您要找的是不是：
第89行：ash 相似度：0.723797 阈值：0.439006
第1512行：she 相似度：0.723797 阈值：0.439006
第1533行：shy 相似度：0.723797 阈值：0.439006
第438行：dish 相似度：0.597631 阈值：0.439006
第604行：fish 相似度：0.597631 阈值：0.439006
第1337行：push 相似度：0.597631 阈值：0.439006
第1437行：rush 相似度：0.597631 阈值：0.439006
第1516行：ship 相似度：0.597631 阈值：0.439006
第1518行：shoe 相似度：0.597631 阈值：0.439006
第1520行：shop 相似度：0.597631 阈值：0.439006
第1526行：shot 相似度：0.597631 阈值：0.439006
第1530行：show 相似度：0.597631 阈值：0.439006
第1532行：shut 相似度：0.597631 阈值：0.439006
第1673行：such 相似度：0.597631 阈值：0.439006
```

图 1（未引入首字母相似机制）

```
C:\Users\Administrator\Documents\源代码\程设大作业\程设...
请输入您想要搜索的单词（不区分大小写）（PS：按俩0退出程序）：
attent
没有找到这个单词！
您要找的是不是：
第100行：Attend 相似度：0.881314 阈值：0.690932
第101行：Attention 相似度：0.738752 阈值：0.690932
第6行：absent 相似度：0.714859 阈值：0.690932
compu
没有找到这个单词！
您要找的是不是：
第328行：computer 相似度：0.666952 阈值：0.659670
numb
没有找到这个单词！
您要找的是不是：
第1144行：number 相似度：0.737423 阈值：0.616667
sh
没有找到这个单词！
您要找的是不是：
第1512行：she 相似度：0.751417 阈值：0.439006
第1533行：shy 相似度：0.751417 阈值：0.439006
第89行：ash 相似度：0.651417 阈值：0.439006
第1516行：ship 相似度：0.637868 阈值：0.439006
第1518行：shoe 相似度：0.637868 阈值：0.439006
第1520行：shop 相似度：0.637868 阈值：0.439006
第1526行：shot 相似度：0.637868 阈值：0.439006
第1530行：show 相似度：0.637868 阈值：0.439006
```

图 2（引入首字母相似机制）¹

¹ 输出超过五个的情况在前面已经说明

```
C:\Users\Administrator\Documents\源代码\程设大作业\程设大作业.exe
请输入您想要搜索的单词（不区分大小写）（PS：按俩0退出程序）：
atten
没有找到这个单词！
您要找的是不是：
第100行: Attend 相似度: 0.874503 阈值: 0.659670
第101行: Attention 相似度: 0.634800 阈值: 0.659670
第1015行: matter 相似度: 0.621216 阈值: 0.659670
第26行: after 相似度: 0.620888 阈值: 0.659670
第6行: absent 相似度: 0.609533 阈值: 0.659670
attent
没有找到这个单词！
您要找的是不是：
第100行: Attend 相似度: 0.881314 阈值: 0.690932
第101行: Attention 相似度: 0.738752 阈值: 0.690932
第6行: absent 相似度: 0.714859 阈值: 0.690932
第1217行: patient 相似度: 0.673249 阈值: 0.690932
第87行: artist 相似度: 0.628788 阈值: 0.690932
operasion
没有找到这个单词！
您要找的是不是：
第1175行: operation 相似度: 0.897727 阈值: 0.750000
第1231行: person 相似度: 0.601349 阈值: 0.750000
第1174行: operate 相似度: 0.568201 阈值: 0.750000
第1176行: opposite 相似度: 0.560005 阈值: 0.750000
第1283行: population 相似度: 0.550906 阈值: 0.750000
```

图 3（未引入阈值机制）

```
C:\Users\Administrator\Documents\源代码\程设大作业\程设大...
请输入您想要搜索的单词（不区分大小写）（PS：按俩0退出程序）：
atten
没有找到这个单词！
您要找的是不是：
第100行: Attend 相似度: 0.874503 阈值: 0.659670
attent
没有找到这个单词！
您要找的是不是：
第100行: Attend 相似度: 0.881314 阈值: 0.690932
第101行: Attention 相似度: 0.738752 阈值: 0.690932
第6行: absent 相似度: 0.714859 阈值: 0.690932
operasion
没有找到这个单词！
您要找的是不是：
第1175行: operation 相似度: 0.897727 阈值: 0.750000
```

图 4（引入阈值机制）

基于以上优化分析，我们的程序主要实现如下功能：

1、主函数实现功能

- ①打开/关闭特定词典文件功能；
- ②用户输入单词功能；
- ③词典单词、输入单词的大小写转换；
- ④单词的精确匹配；
- ⑤文件的读入、输出日志；

2、子函数实现功能

- ①biseach 函数实现在词典中的二分查找；
- ②checkNonzero 函数；
- ③sum_up 函数；
- ④compute_cos 函数实现特征向量夹角的余弦值；
- ⑤editDistance 函数实现编辑距离的计算；
- ⑥threshold 函数实现阈值的确定；
- ⑦display 函数实现结果的按链表输出。

六、感想体会

林子祺（主要负责：整体优化、用特征向量方法计算相似度）

感想：这次大作业费了很多心思，最后的测试结果比较精确，还是挺欣慰的。课程项目让我学会了很多，写一个大程序（对新手来说）能让自己思维更严谨，整个框架的逻辑性比平时的小程序更强。分析问题和解决问题的实践在将来的实际应用中也很宝贵，希望自己能在实战道路上越走越远。

宋晓林（主要负责：文件读入、日志输出）

感想：这次有关模糊查询的程设作业中我负责的是文件的读入和输出部分，由于在之前并没有接触过有关文件的内容，因此在一开始也是费了点脑筋去理解程序的实现过程，在实现了读入后考虑到相关需求又改为读入指定文件并将结果输出。感觉最大的收获还是对于文件相关的学习以及实现目标后的愉悦。

杨国兴（主要负责：编辑距离函数）

感想：这次编写编辑距离的函数巩固了这学期学到的递归知识，也学会了小组合作完成程序，从其他组员特别是组长那里学到了许多知识。

杨敬栋（主要负责：输出部分）

感想：这次编写排序输出加深了我对排序方面的理解，平时总是用冒泡减少了对其他算法的应用，这次用的方法让我再次感受到合作的重要和对算法的感悟吧

叶挺茂（主要负责：精确匹配）

感想：这次大作业实践，一开始拿到题目的时候觉得是个非常复杂的项目。但是当把这个课题细化之后实际上问题非常明显。再加上组内合理的分工，这次大作业完成得比较顺利。初期完成后，经过老师指导又在程序上有了更近一层的、更好的功能。总的来说这是对编程技能的一项考验也是一次提高。

附录 1、源代码

```
#include<stdio.h>
#include<string.h>
#include<windows.h>
#include <math.h>

struct simWord
{
    int line;
    char word[32];
    double sim;
    double threshold;
    struct simWord *next;
};

int bisearch( char ch[1997][32], int low,int high,char key[32] );//精确匹配时使用二分查找
int checkNonzero(double *tmp1,double *tmp2);//计算非零项的个数
double sum_up(double *ary,int n);//求数组和
double compute_cos(double *ary1,double *ary2,int n);//根据字母出现频率计算相似度
double editDistance(char *p,char *q);//根据编辑距离计算相似度
int min_of_three(int a,int b,int c);//三者中取最小值
double threshold(double n);//根据用户输入单词长度确定输出阈值
void display(struct simWord* head);//将相似度达标的单词组成的有序链表输出
FILE *fp, *stream; //定义文件指针

int main()
{
    double tmp1[26],tmp2[26];
    double accordance[1997],initial_sim;
    int i = 1 , count = 0, j = 0, k = 0;
    char dic[1997][32], ch[1997][32], key[32], quit[2]={48,48},
    filename[200], filename2[200];
```

```

    struct simWord *head, *prv, *latter, *cur;
    printf("请您输入您想打开的文件路径（注意文件类型）：\n");
    scanf("%s", &filename);
    fp = fopen(filename, "r");
    printf("请您输入想要将输出保存的文件：\n");
    scanf("%s", &filename2);
    stream = fopen(filename2, "w");
    printf("请输入您想要搜索的单词（不区分大小写）（PS：输入两个 0 退出程序）：\n");
    if(fp == NULL) //提示文件是否打开成功
    {
        printf("文件打开失败!\n");
        fprintf(stream, "文件打开失败!\n");
        exit(1);
    }
    for(k = 0; k < 1997; k++)
    {
        fgets(dic[k], 32, fp); //调用 fgets 函数导入单词表单词
        strcpy(ch[k], dic[k]);
        for (i = 0; i < 32; i++) //将单词表导入单词统一转化为小写
        {
            if ( (dic[k][i] < 65 && dic[k][i] != 39) || (dic[k][i] >
90 && dic[k][i] < 97) || (dic[k][i] > 122) ) //若读入其他字符如默认存在的
换行符则置为空字符
                dic[k][i] = 0;
            if (ch[k][i] >= 'A' && ch[k][i] <= 'Z')
                ch[k][i] = ch[k][i] - 'A' + 'a';
            if ( (ch[k][i] < 65 && ch[k][i] != 39) || (ch[k][i] >
90 && ch[k][i] < 97) || (ch[k][i] > 122) ) //若读入其他字符如默认存在的
换行符则置为空字符
                ch[k][i] = 0;
        }
    }
    while(1)
    {
        head = NULL;
        count = 0;
        scanf("%s", key); //键盘输入欲查单词
        if(strcmp(key, quit) != 0)
        {
            fprintf(stream, "%s", key);
            for (i = 0; i < strlen(key) ; i++) //将输入单词转化为小
写
                {

```

```

        if (key[i] >= 'A' && key[i] <= 'Z')
            key[i] = key[i] - 'A' + 'a';
        if ( (key[i] < 65&&key[i] != 39) || (key[i] >
90&&key[i] < 97) || (key[i] > 122) )//若读入其他字符如默认存在的换行
符则置为空字符
        {
            printf("请勿输入非字母字符\n");
            fprintf(stream, "请勿输入非字母字符\n");
            count++;
            break;
        }
    }
    if(bisearch(ch, 0, 1996, key) >= 0)
    {
        printf("所在行: %d\n", bisearch(ch, 0, 1996, key)+1 );
        fprintf(stream, "所在行: %d\n", bisearch(ch, 0, 1996,
key)+1 );
        count++;
    }//用二分查找找到用户输入的单词
    if(count == 0) //两个字符串不同则进行下一步模糊查找
    {
        printf("没有找到这个单词!\n");
        fprintf(stream, "没有找到这个单词!\n");
        for(k = 0; k < 1997; k++)
        {
            for(i = 0; i < 26; i++)
            {
                tmp1[i] = 0;
                tmp2[i] = 0;
            }
            for(i = 0; i < strlen(key); i++)
            {
                if(key[i] >= 97 && key[i] <= 122)
                    tmp1[key[i]-97]++;
            }
            for(i = 0; i < strlen(ch[k]); i++)
            {
                if(ch[k][i] >= 97 && ch[k][i] <= 122)
                    tmp2[ch[k][i]-97]++;
            } //统计字母数
            int len = checkNonzero(tmp1, tmp2);
            double input_word_vector[len],
aim_word_vector[len];
            i = 0;

```

```

        for(j = 0; j < 26; j++)
        {
            if(!tmp1[j] && !tmp2[j])continue;
            else if( (tmp1[j] || tmp2[j]) && i<len)
            {
                input_word_vector[i] = tmp1[j];
                aim_word_vector[i] = tmp2[j];
                i++;
            }
        }//只保留两个向量至少一个存在非零项的维度
        double sum1 = sum_up(input_word_vector, len),
sum2 = sum_up(aim_word_vector, len);
        for(i = 0; i < len; i++)
        {
            input_word_vector[i] /= sum1;
            aim_word_vector[i] /= sum2;
        }//归一化处理
        double length = (double)strlen(key);
        double root = sqrt(length);
        if( ch[k][0] == key[0] )
            initial_sim = 0.1;
        else
            initial_sim = 0.0;//首字母相同的单词会被认为更
相似
        accordance[k] = compute_cos(input_word_vector,
aim_word_vector, len)*(0.9/root) + editDistance(key, ch[k])*(0.9-
0.9/root) + initial_sim;
        //根据单词长度对三种方法求得的相似度取加权平均值
        if(accordance[k] >= threshold(length) )
        { //如果相似度大于阈值则建立达标单词链表
            cur = (struct simWord*) malloc(sizeof (struct
simWord) );

            cur->line = k+1;
            strcpy(cur->word, dic[k]);
            cur->sim = accordance[k];
            cur->threshold = threshold(length);
            cur->next = NULL;
            if(head == NULL)
                head = cur;
            else
            {
                prv = NULL;
                latter = head;

```

```

        while(latter != NULL && latter->sim >=
cur->sim)
        {
            prv = latter;
            latter = latter->next;
        }
        cur->next = latter;
        if(prv == NULL)
            head = cur;
        else
            prv->next = cur;
    }
}
} //根据相似度从大到小按顺序接入链表，当相似度相等时按
字典序接入
    display(head); //输出链表
}
}
else break;
}
fclose(fp);
fclose(stream); //关闭文件
return 0;
}

```

```

int bisearch( char ch[1997][32], int low, int high, char key[32] )
{
    int mid;
    mid=(low+high)/2;
    if(low>high)
        return -1;
    if(strcmp(ch[mid], key) == 0)
        return mid;
    else if(strcmp(ch[mid], key) > 0)
        return bisearch(ch, low, mid-1, key);
    else
        return bisearch(ch, mid+1, high, key);
}

```

```

int checkNonzero(double *tmp1, double *tmp2)
{
    int sum = 0, i;

```

```

    for(i = 0; i < 26; i++) {
        if( tmp1[i] || tmp2[i] )
            sum++;
    }
    return sum;
}

double sum_up(double *ary, int n)
{
    int i;
    double sum = 0.0;
    for(i = 0; i < n; i++) {
        sum += ary[i];
    }
    return sum;
}

double compute_cos(double *ary1, double *ary2, int n)
{
    double quasum1 = 0.0, quasum2 = 0.0, quasumOfDif = 0.0, temp;
    double dis1, dis2, dis, cos;
    int i;
    for(i = 0; i < n; i++) {
        quasum1 += (ary1[i] * ary1[i]);
        quasum2 += (ary2[i] * ary2[i]);
        temp = fabs(ary1[i] - ary2[i]);
        quasumOfDif += (temp * temp);
    }
    dis1 = sqrt(quasum1);
    dis2 = sqrt(quasum2);
    dis = sqrt(quasumOfDif);
    cos = (dis1*dis1 + dis2*dis2 - dis*dis)/(2 * dis1 * dis2);
    return cos;
}

double editDistance(char *p, char *q) {
    int i, j;
    int len1 = strlen(p); //统计输入的单词长度
    int len2 = strlen(q); //统计对比单词的长度
    int step[len1+1][len2+1]; // 建立一个矩阵
    step[0][0] = 0;
    for(i=1; i <= len1; i++)
    {
        step[i][0] = i;

```



```

    }
    for(j=1; j <= len2; j++)
    {
        step[0][j] = j;
    }
    //限定边界的值
    int k,l;
    for(k=1; k <= len1; k++)
    {
        for(l=1; l <= len2; l++)
        {
            if(*(p+k-1) == *(q+l-1))
            {
                step[k][l]=step[k-1][l-1];
            }
            else
            {
                int min;
                min = min_of_three(step[k-1][l-1], step[k-
1][l], step[k][l-1]);
                step[k][l] = min+1;
            }
        }
    }
    //逐个填矩阵
    double edit_dis = (double)step[len1][len2]/(double)(len1/2 +
len2/2); //算编辑距离
    double result = 1-edit_dis;
    return result;
}

int min_of_three(int a,int b,int c)
{
    int min;
    if(b <= a&&b <= c)min=b;
    if(a <= b&&a <= c)min=a;
    if(c <= b&&c <= a)min=c;
    return min;
}

double threshold(double n)
{
    double root = sqrt(n);

```

```

    double dis1 = sqrt( ( 1/(n*n) ) * (n-
root/2) ),dis2=sqrt( (1/(n*n) ) * (n-root/2) ),dis=sqrt( (1/(n*n) ) *
(root/2) );
    double cos = (dis1*dis1+dis2*dis2-dis*dis) / (2 * dis1 * dis2);
    double sim = (n - 1.2*root)/n,result = 0.5*cos + 0.5*sim;
    return result;
}

void display(struct simWord* head){
    struct simWord *cur = head;
    int i = 0;
    while( (cur != NULL&& i < 5) || ( cur != NULL&&cur->next->next !=
NULL && cur->next !=NULL &&cur->next->next->sim == cur->next->sim) )
    {
        if(!i)
        {
            printf("您要找的是不是: \n");
            fprintf(stream, "您要找的是不是: \n");
        }
        printf("第%d 行: %s 相似度: %lf 阈值: %lf\n", cur->line,
cur->word, cur->sim, cur->threshold);
        fprintf(stream, "第%d 行: %s 相似度: %lf 阈值: %lf\n",
cur->line, cur->word, cur->sim, cur->threshold);
        cur = cur->next;
        i++;
    }
}

```

附录 2、参考资料

[1]E. Garcia, PhD, Cosine Similarity Tutorial,04-10-2015;

[2]谭浩强,C 程序设计,清华大学出版社;

[3]<http://www.dreamxu.com/books/dsa/dp/edit-distance.html>

[4]https://en.wikipedia.org/wiki/Edit_distance