C语言附加题-单词纠错

由于发音相近或者记忆不准确，我们日常生活中常有单词拼写错误的情况发生。而对于自然语言处理，一般以词为基本处理单元。因此单词纠错是自然语言处理重要的基础算法之一。单词纠错的方法通常是检测未出现在词典中的单词，然后借助于单词之间的**编辑距离**来从词典中寻找正确的单词。

> **莱文斯坦距离**，又称Levenshtein距离，是编辑距离的一种。指两个字串之间，由一个转成另一个所需的最少编辑操作次数。允许的编辑操作包括将一个字符替换成另一个字符，插入一个字符，删除一个字符。
>
> 例如将kitten一字转成sitting：
> sitten （k→s）
> sittin （e→i）
> sitting （→g）
>
> 除单词纠错之外，编辑距离还应用于DNA分析、语音辨识、抄袭侦测问题中。

Levenshtein编辑算法基本原理：首先，假设对于字符串A和B，我们使用一个二维数组distance[1+length of A][1+length of B]，其中distance[i][j]的值表示将 A的1到i部分 转换为 B的1到j部分 需要的最少编辑次数。

所以，在i等于0时，也就是说对于distance[0,j]，A的空子串部分到B的前j个元素组成的子串的转换，需要j此操作。同理可得distance数组的首行和首列为0到i和0到j。
对于剩余的distance[i][j]（也就是A1~i元素的子串和B1~j元素的子串的距离），可以由distance[i-1][j]，distance[i][j-1]，distance[i-1][j-1]来确定，如果计算A[i]和B[j]相同，取min（distance[i-1][j]+1，distance[i][j-1]+1，distance[i-1][j-1]）即可。同理，如果计算A[i]和B[j]不同，就要取min（distance[i-1][j]+1，distance[i][j-1]+1，distance[i-1][j-1]+1）。
计算出所有distance元素后，我们要求的A到B的编辑距离就是distance[1+length of A][1+length of B]

举个例子，对于luck和 uck两个字符串计算编辑距离，最终的二维数组计算如下（红色为被min()选中的数值）：

> 字符串ABCD和BCD的距离计算过程：
>
_	**_**	***u***	***c***	***k***
_	**0**	1	2	3
l	1	**1**	2	3
u	2	**1**	2	3
c	3	2	**1**	2

k	4	3	2	**1**

学完C语言的你，是否可以使用编程技巧来对文档中的单词拼写错误进行纠正呢?

题目要求：

读取给定文档(words.txt)和单词表(vocabulary.txt)，利用单词表找出文档中拼写错误的单词并将其改正，把正确的内容存到新的文档之中，命名为words_correct.txt。

为了保证数据的真实性，中间可能夹杂多余空格或者中文等不合格式的信息，中文字符删除，行输出按给定的模板格式输出；编辑距离相同的词可以取首个出现的词，鼓励加入其它判断进一步筛选，如是否可通过交换纠正(raed->read)

示例：

> 0351 grandmother kind/**mather**/孙子/grandaughter
> 0707 London England/**Endlish**/Cambridge developed

要更改为

> 0351 grandmother kind/**mother**/grandaughter
> 0707 London England/**English**/Cambridge/developed

数据格式：

输出文件每一行分别为：行号 单词1 单词2/单词3/…/单词n