CSAPP第一次作业实验报告

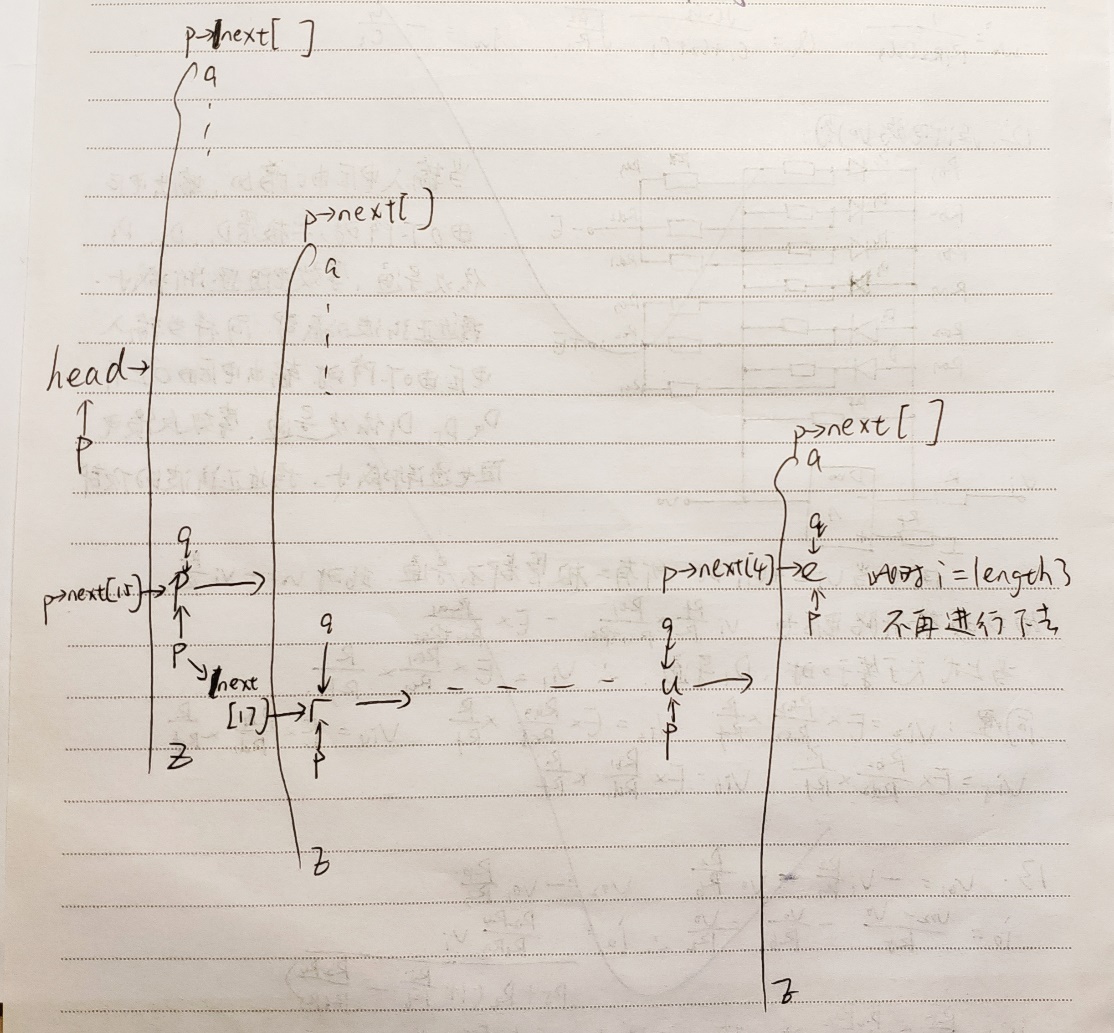
张宇宸 16307130046

1. 思路

题目要求从txt文件中读取并统计单词词频，完成后输出到另一个外部txt文件中。在这种不知道一共会有多少个单词输入的时候，应该有两种处理方式：一是开辟动态存储空间，比如应用链表或者长度可变的数组；二是定义一个足够大的数组来尽可能地装入所有单词。这里我选择了后者。

我总体的思路是：先从文件一个一个字符读入，读入的同时判断是否是与单词有关的字符，在本题要求下只有4种字符有效：大写字母，小写字母，缩写符（'），连字符（-），并变成小写后存入暂存数组。在读到和单词无关的字符时，判断是否是一个单词的结束标志，然后结束该暂存数组，开始进行单词处理工作。首先把这个单词里的缩写符和连字符都去掉，然后按照字母树来储存单词或增加词频数。所有单词都处理结束后将字母树里的单词还原，按照词频进行排序，最后输出，打印运行时间。下面是各部分的思路。

1. 读入和判断单词：可以在读入单个字符之后进行if判断，是这4种所需字符就存入暂存字符数组里，不是的话就先进行判断，如果tempword是空的，则它的第一位就应该是‘\0’，这意味着不需要进行后面有关单词的处理了。否则就要给tempword的最后一位写入’\0’，并用字母树处理。
2. 去除’和-：遍历该寄存数组，在发现这两个符号的时候，把这个位置只有的所有字符都向前移动一位，即可覆盖掉’或-。需注意的是，最好能判断出最后一位在移动的情况并在它后面补一个’\0’。
3. 将大写字母变成小写字母：可以在读入时就进行修改，如果判断该字符是大于A小于Z的字符，查ASCII码表发现所有大写字母码值都比对应的小写字母码值小32，因此可以直接将其加32变成小写。
4. 用字母树的方式存储数据：以二叉树为原型的扩展。我们可以用一个头指针发散出一个长度为26的数组，用来寄存26个英文字母，这26个字母继续发散出26\*26个字母，如此继续下去，可以形成一个包含所有可能单词的字母树。当读取寄存数组时，应该逐位读取并储存。这里我用字母与a的距离来作为数组的下标，即该字母的位置。如果某个字母没有在next[location]里的话，说明以前从来没有出现过这样一个单词，否则在对应的next[]里一定会有这个字母。

下面我举一些例子来说明这种方法具体的工作模式。当第一次读入单词prologue时（见下图），由于之前什么单词也没有读入过，p->next[26]的任何一处都是NULL的，因此须开创一个新的空的区域来将第一个字母“p”存入，得知p与a的距离是15，因此p->next[15]就与q连接上，然后p跳转到当前q指针的位置（原p->next[15]）上。对于下一个字母“r”，它与a的距离是17，因为q->next[17]是NULL的，因此和刚才一样，开创一个空的区域来连接到q->next[17]上，p再向下走一位到当前q的位置（原p->next[17]）上。如此继续进行下去，因为循环条件是变量i小于字符串长度，因此在查询到最后一位并将其储存后将会停止循环，并将此时的词频加一。这就是从未出现过的单词的存储方式。

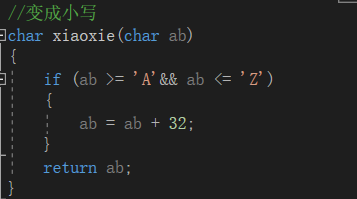
那么对于部分出现过的单词是如何存储的呢？例如bet和bee，当已经存在bet时，再读到bee，第一个字母b是已经存在的，因此可以直接跳到下一位，而不需再开创新空间q；第二位e也是已经存在的，跳到第三位。而如果以前从未出现过以“bee”为开头的单词，则第三位在e的位置，即p->next[4]一定是NULL，因此开辟一个新的空间q将其存入；即使以前存在过“bee”开头的单词，因为长度不同，第三位在e处就结束了，因此也可以判断出是一个新的单词存入。只有在长度和一连串的next[]都相同的情况下，才能认为是同一个单词，于是对应位置的p->num加一。

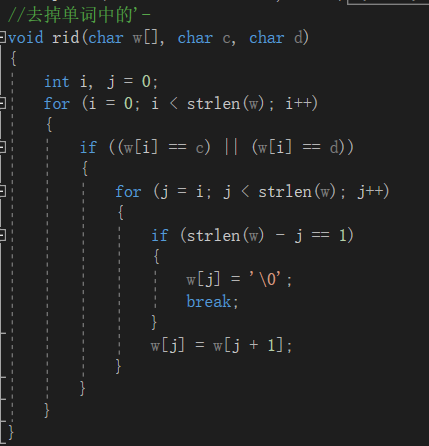
对于同一单词的多次出现，逐位查找发现每次的p->next[]上都非NULL，即有对应字母出现过，所以不会开q空间。当单词的每个字母都被查询过后，由于始终没有开创新空间，所以整个字母树结构没有任何变化，只有最后的p->num加了1，证明这个单词重复出现一次，并将其体现在词频+1上。

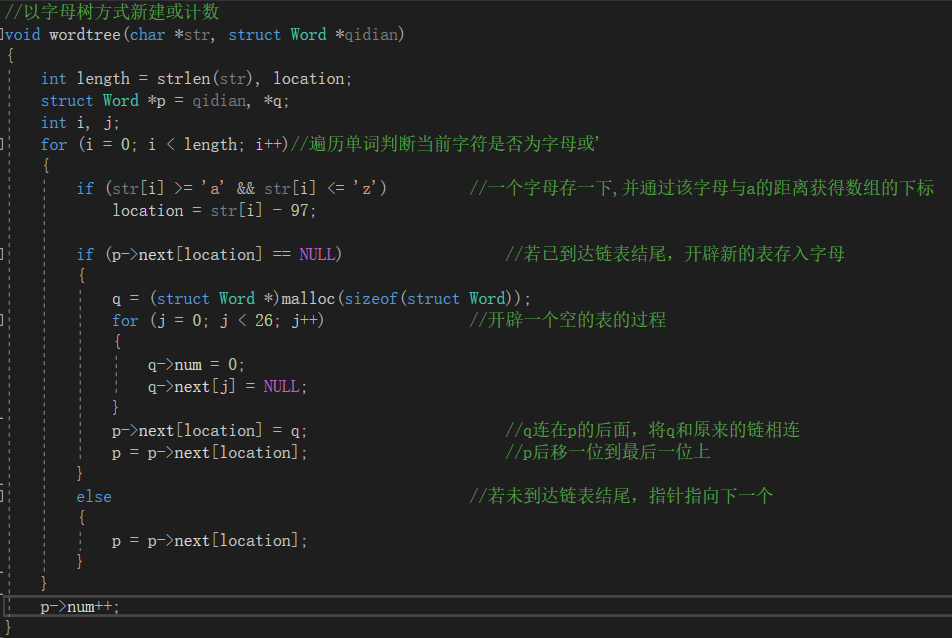
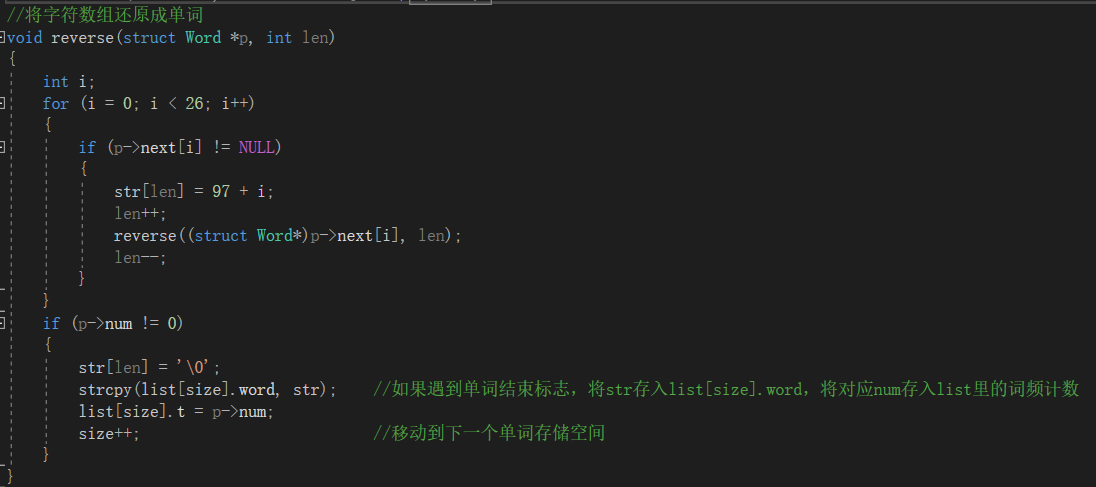
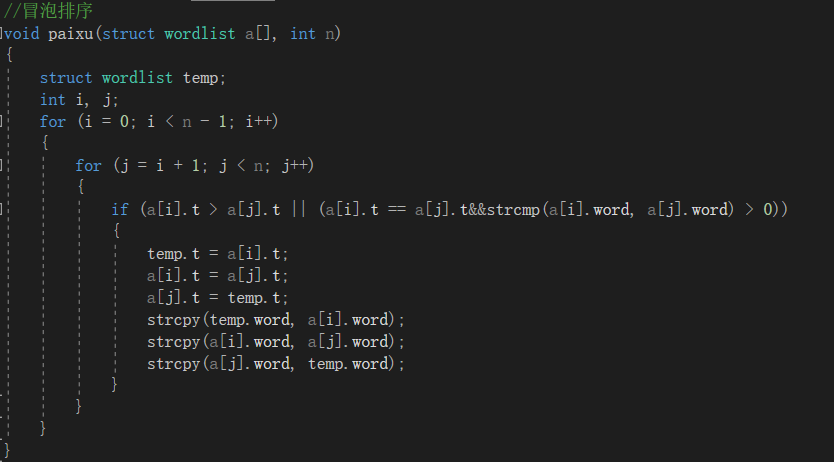
对于每一个单词，都进行一次这样的存储过程。当所有字符都读完后，所有单词都将暂时以其处在字母树的不同位置的形式被记录下来，以及它们的词频。

1. 还原单词：因为存储单词时将英文字母进行处理过，它们不再是字母的形式了，所以需要用一个函数来从后往前地把每一处有值的位置变成字母的样子，这个过程是字母树的反向的收束的过程。但是因为不可能直接从字母树的最后处向前查询（因为不知道最长的单词是什么），所以要用嵌套来实现。为了实现将字母又一一存进字符数组变成一个字符串，可以用当初把字符读入时的方法，即str[len]=字符，len++。因为是倒着读取字母树的，所以还要辅以len - -来实现倒着储存字符。

当一个单词还原完后，如果它的词频不是0，则将这个单词和词频存入最开始创建的100万容量的单词表里。这里我在最开始犯了个错误，即没有判断词频是否是0，然后就会出现一个单词的所有组成部分都被输出的错误，并且其词频是0。比如abondon，除了abondon词频1，a在其他部分有被读取过所以词频不是0之外，还会输出ab，abo，abon，abond，abondo这些不是单词的字符串，词频均为0。

1. 对单词词频进行排序：这里我采用了冒泡排序，一是因为很容易实现，不易出错，二是因为效率较高，最坏时间复杂度O(n²)，最好时间复杂度O(n)，且足够稳定。同时因为快排的时间复杂度要优于冒泡排序，我也尝试了用快速排序，可惜最后运行发现时间比冒泡还要长，推测原因有二：一是我的代码水平不够熟练，写的比较复杂；二是快速排序不是很稳定。因此在几种较为稳定的排序方法（直接插入排序，冒泡排序，归并排序等）里，冒泡排序属于在我能力范围内的最好选择。鉴于冒泡排序十分普遍，思路简单，我将在下面直接贴出算法。
2. 输出结果：遍历list字母表，逐个输出到文件里。并通过用结束时的clock减去开始时的clock即可得出程序运行的时间。
3. 算法
4. 



1. 
2. 
3. 
4. 耗时

运行10次（单位：ms）

11698 11636 11618 11609 11601 11611 11646 11695 11650 11689

平均时长：

11645

于刚开机情况下测得。（在VS2017里平均是7453ms）

四、运行环境

Dev C++ 5.11

TDM-GCC 4.9.2 64-bit Release

Windows10 64-bit

（在有些机器上在Dev C++里编译运行可能会提示“源文件未编译”，建立一个新的空项目把代码复制进去就可以编译运行了。原因不明）

五、实验总结

最开始我在写程序的时候使用了链表的方式来存储单词，在读取较小的文件（1M左右）结果正常输出，但换成原始input.txt后每次运行都失败，改用VS后提示遇到断点，得知是内存溢出了。因为自身代码能力和对内存方面的了解不够深入，改了两天也没能成功，所以换成了这种先定义大数组的办法。在大一上学习C语言后，我没有再使用过，因此这次程序设计比较生疏。我此次使用的算法都比较粗暴，代码也没有很简洁，这些应该都是导致运行时间较长的因素。在这次练习过程中，我也意识到真正的程序是可以处理像input.txt这样巨大的文件的，而不是对随便一个小样本能实现功能就可以了。要想高效、正确地实现要求，对计算机系统的学习是十分必要的。