# 代码性能实验实验报告

作者：yyww322

实验目的: 本实验通过对代码性能的分析和优化，使学生理解和掌握课程中关于代码性能的相关内容，努力养成注重代码性能的编程习惯

## 功能实现:

本实验要求实现对一个大容量英文文献的分词与索引建立，并能接收对该文献内容的全文单词检索功能。更具体地，对于一个含有以分隔符（逗号，空格，分号等非英文字母）分隔开的N个单词的文本文献（其中单词可能重复），程序对这N个单词进行读入，并建立相应的索引结构。该索引结构能够支持对任意单词的查询，返回结果为该单词在文献中出现的所有位置信息（以单词出现的顺序号标记位置）。

本实验完成度至少满足以下要求：

支持5M大小的文本输入

支持超过23000 种不同的单词输入

15分钟内的文件读取与处理时间

1秒内的检索返回时间

## 实验过程:

按照实验要求,分别实现了其所需的功能,并采用大小为1.7M的英文小说进行了实验。

首先实现wordbase.py，跟据题目要求需要由用户输入指定读取文件名称，此处额外扩展了可以指定存储字典的文件名称，如果用户不输入字典名称，则默认存储在my\_dictionary.pk1文件中。

读取文件使用了try…exception的格式，用来避免程序崩溃。

读取出所有文件中的字段后，首先采用传统的循环模式逐字对字段分析，在一次循环内完成分词并存入字典中。此后采用正则表达式的方式重新做了一次分词存储，对两次结果进行了比较。

存储字典采用了pickle功能，直接存储整个字典，方便存取功能的开发。

此后实现了wordquery.py，此功能比较简单，具有两个输入参数，第一个参数代表了查询单词的文件名，第二个参数可以指定存储字典的名称，也可以不输入，默认为my\_dictionary.pk1。

同样的读取过程采用了try…exception格式避免未知错误发生。此后就是简单的检索功能，并且显示上面按照要求进行了一些文字处理。

## 实验结果与分析：

由于wordquery.py实际的执行时间主要为从硬盘读取文件到内存的时间，读入内存后，python字典的查找复杂度为o(N)，基本可以忽略不计，所以这里不作分析了，主要对比一下自己写的程序和使用正则表达式实现的wordbase.py的区别：

首先是自己写的分词存储部分：

for i in range(len(self.my\_str)):

if 'a' <= self.my\_str[i] <= 'z':

word += self.my\_str[i]

else:

if 'Z' >= self.my\_str[i] >= 'A':

word += self.my\_str[i].lower()

else:

if len(word) >= 1:

self.save\_directionary(word, word\_count)

word\_count += 1

word = ''

if len(word) >= 1:

self.save\_directionary(word, word\_count)

self.word\_count = word\_count

从头执行到尾，如果有a-z或者A-Z中间的值则认为是单词的一部分，如果有别的符号则认为单词结束，此处应分词，将单词通过调用函数存入字典变量中。下面是使用profile调用的输出结果：

原始程序执行时间:

Total Vocabulary: 19436

Total Words: 295475

1304847 function calls in 9.048 seconds

Ordered by: standard name

ncalls tottime percall cumtime percall filename:lineno(function)

1 0.000 0.000 0.000 0.000 :0(\_getdefaultlocale)

295474 0.562 0.000 0.562 0.000 :0(append)

2 0.000 0.000 0.000 0.000 :0(close)

1 0.094 0.094 0.094 0.094 :0(dump)

1 0.000 0.000 9.048 9.048 :0(exec)

664972 1.560 0.000 1.560 0.000 :0(len)

48907 0.109 0.000 0.109 0.000 :0(lower)

2 0.000 0.000 0.000 0.000 :0(open)

2 0.000 0.000 0.000 0.000 :0(print)

1 0.047 0.047 0.047 0.047 :0(read)

1 0.000 0.000 0.000 0.000 :0(setprofile)

1 0.016 0.016 9.048 9.048 <string>:1(<module>)

1 0.000 0.000 0.000 0.000 index\_generator.py:11(\_\_init\_\_)

1 0.000 0.000 0.047 0.047 index\_generator.py:18(get\_str)

1 4.103 4.103 8.892 8.892 index\_generator.py:30(gen\_directionary)

295474 2.558 0.000 3.760 0.000 index\_generator.py:55(save\_directionary)

1 0.000 0.000 0.094 0.094 index\_generator.py:63(save\_dir\_to\_file)

1 0.000 0.000 9.032 9.032 index\_generator.py:75(begin\_sort)

1 0.000 0.000 0.000 0.000 locale.py:587(getpreferredencoding)

1 0.000 0.000 9.048 9.048 profile:0(main(sys.argv[1:]))

0 0.000 0.000 profile:0(profiler)

1 0.000 0.000 9.032 9.032 wordbase.py:11(main)

Process finished with exit code 0

可以看到len这里稍微执行的次数多了一些，由于加了一个是否长度大于1的判断，加此判断的初衷是如果不想要单字的字母做为单词日后可以修改。

下面是使用正则表达式实现同样功能的代码：

words = re.split('[^a-zA-Z]+', self.my\_str)

for word in words:

self.save\_directionary(word.lower(), self.word\_count)

self.word\_count += 1

使用正则表达式比较简洁，但是也不容易看懂，下面是profile产生的结果:

E:\1\Python33\python.exe E:/1/Python33/mywork/directionary/wordbase.py test1.txt

Total Vocabulary: 19436

Total Words: 295476

1182084 function calls (1182081 primitive calls) in 6.240 seconds

Ordered by: standard name

ncalls tottime percall cumtime percall filename:lineno(function)

1 0.000 0.000 0.000 0.000 :0(\_getdefaultlocale)

295505 0.484 0.000 0.484 0.000 :0(append)

2 0.016 0.008 0.016 0.008 :0(close)

1 0.000 0.000 0.000 0.000 :0(compile)

1 0.078 0.078 0.078 0.078 :0(dump)

1 0.000 0.000 6.240 6.240 :0(exec)

11 0.000 0.000 0.000 0.000 :0(isinstance)

1 0.000 0.000 0.000 0.000 :0(items)

295518/295517 0.686 0.000 0.686 0.000 :0(len)

295475 0.608 0.000 0.608 0.000 :0(lower)

4 0.000 0.000 0.000 0.000 :0(min)

2 0.000 0.000 0.000 0.000 :0(open)

4 0.000 0.000 0.000 0.000 :0(ord)

2 0.000 0.000 0.000 0.000 :0(print)

1 0.031 0.031 0.031 0.031 :0(read)

1 0.000 0.000 0.000 0.000 :0(setprofile)

1 0.281 0.281 0.281 0.281 :0(split)

1 0.031 0.031 6.240 6.240 <string>:1(<module>)

1 0.000 0.000 0.000 0.000 index\_generator.py:12(\_\_init\_\_)

1 0.000 0.000 0.031 0.031 index\_generator.py:19(get\_str)

1 1.778 1.778 6.068 6.068 index\_generator.py:31(gen\_directionary)

295475 2.231 0.000 3.401 0.000 index\_generator.py:62(save\_directionary)

1 0.000 0.000 0.094 0.094 index\_generator.py:70(save\_dir\_to\_file)

1 0.016 0.016 6.209 6.209 index\_generator.py:82(begin\_sort)

1 0.000 0.000 0.000 0.000 locale.py:587(getpreferredencoding)

1 0.000 0.000 6.240 6.240 profile:0(main(sys.argv[1:]))

0 0.000 0.000 profile:0(profiler)

1 0.000 0.000 0.281 0.281 re.py:183(split)

1 0.000 0.000 0.000 0.000 re.py:268(\_compile)

1 0.000 0.000 0.000 0.000 sre\_compile.py:178(\_compile\_charset)

1 0.000 0.000 0.000 0.000 sre\_compile.py:207(\_optimize\_charset)

8 0.000 0.000 0.000 0.000 sre\_compile.py:24(\_identityfunction)

2/1 0.000 0.000 0.000 0.000 sre\_compile.py:32(\_compile)

1 0.000 0.000 0.000 0.000 sre\_compile.py:357(\_simple)

1 0.000 0.000 0.000 0.000 sre\_compile.py:362(\_compile\_info)

2 0.000 0.000 0.000 0.000 sre\_compile.py:468(isstring)

1 0.000 0.000 0.000 0.000 sre\_compile.py:471(\_code)

1 0.000 0.000 0.000 0.000 sre\_compile.py:486(compile)

3 0.000 0.000 0.000 0.000 sre\_parse.py:127(\_\_len\_\_)

6 0.000 0.000 0.000 0.000 sre\_parse.py:131(\_\_getitem\_\_)

1 0.000 0.000 0.000 0.000 sre\_parse.py:135(\_\_setitem\_\_)

1 0.000 0.000 0.000 0.000 sre\_parse.py:139(append)

3/2 0.000 0.000 0.000 0.000 sre\_parse.py:141(getwidth)

1 0.000 0.000 0.000 0.000 sre\_parse.py:179(\_\_init\_\_)

13 0.000 0.000 0.000 0.000 sre\_parse.py:184(\_\_next)

5 0.000 0.000 0.000 0.000 sre\_parse.py:203(match)

9 0.000 0.000 0.000 0.000 sre\_parse.py:209(get)

1 0.000 0.000 0.000 0.000 sre\_parse.py:352(\_parse\_sub)

1 0.000 0.000 0.000 0.000 sre\_parse.py:430(\_parse)

1 0.000 0.000 0.000 0.000 sre\_parse.py:68(\_\_init\_\_)

1 0.000 0.000 0.000 0.000 sre\_parse.py:725(fix\_flags)

1 0.000 0.000 0.000 0.000 sre\_parse.py:737(parse)

2 0.000 0.000 0.000 0.000 sre\_parse.py:91(\_\_init\_\_)

1 0.000 0.000 6.209 6.209 wordbase.py:11(main)

Process finished with exit code 0

可以看出，时间只用了6.24秒，比此前的9.048秒快了不少，这应该是python内部对于正则表达式进行了优化导致的。