**Python 实验二**

**学号：**E11414061 **姓名：**曹前松

**实验一：**

密码管理模块运行结果

import time

import random

import pickle

import os

class Diction:

baidupwd={'cqs':'4682'}

\_\_database={'www.baidu.com':baidupwd}

def \_\_init\_\_(self):

if(os.path.exists('database.dat')):

input = open('database.dat','rb')

try:

self.\_\_database=pickle.load(input)

print("初始化载入存档数据成功 "+time.strftime("%Y-%m-%d %H:%M:%S", time.localtime()))

except:

print("初始化载入文件异常")

finally:

input.close()

else:

print("当前数据存档为空")

#self.\_\_database['@time']=time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())

print("you can add('www.\*\*\*.com','acount','password') to add password to database")

print(self.\_\_database) #仅供调试调用。

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

def filecheck(self):

if(os.path.exists('database.dat')):

return True

else:

return False

def add(self,key,subkey,value):

b=self.\_\_database.copy()#以防出现RuntimeError: dictionary changed size during iteratio

for item in b.items():

if key in list(item):

self.\_\_database[key][subkey]=value

else:

temp={subkey:value}

self.\_\_database[key]=temp

def update(self,key,subkey,value):

a = False

b=self.\_\_database.copy()#以防出现RuntimeError: dictionary changed size during iteratio

for item in b.items():

if key in list(item):

self.\_\_database[key][subkey]=value

a = True

if(not a):

print("error don't have this item:|address|:"+key+' |account|: '+subkey+" |yet please add this and retry")

else:

print('update success!(更改数据成功）')

def search(self,key):

b=self.\_\_database.copy()#以防出现RuntimeError: dictionary changed size during iteratio

for item in b.items():

if key in list(item):

print(item)

#使用pickle模块将数据对象保存到文件

def save(self):

output = open('database.dat','wb')

try:

pickle.dump(self.\_\_database,output)

print("写入数据成功")

except:

print("写入文件异常")

finally:

output.close()

#pickle write tp file

#利用pickle模块从文件中重构python对象

def load(self):

input = open('database.dat','rb')

try:

self.\_\_database=pickle.load(input)

print("载入数据成功")

except:

print("读入入文件异常")

finally:

input.close()

def showAll(self):

#添加一个页面元素控制

for item in self.\_\_database.items():

print(item)

print('-------------------------------')

def review(self,num=3):

words=list(self.\_\_database.keys())

for i in range(num):

word=random.choice(words)

print(word)

while(True):

option=input("check:c or skip:s ")

if option.startswith('c') or option.startswith('s'):

break

else:

print("argument error")

if option.startswith('c'):

print( word,':',self.\_\_database[word])

#module 测试模块

if \_\_name\_\_=='\_\_main\_\_':

d= Diction()

d.add('www.baidu.com','test','123456')

#d.add('check','查看')

d.showAll()

d.add('www.baidu.com','test','88888')

d.showAll()

d.add('www.google.com','test','123456')

d.showAll()

d.update('www.google.com','test','modify')

d.update('www.gogle.com','test','modify')

d.showAll()

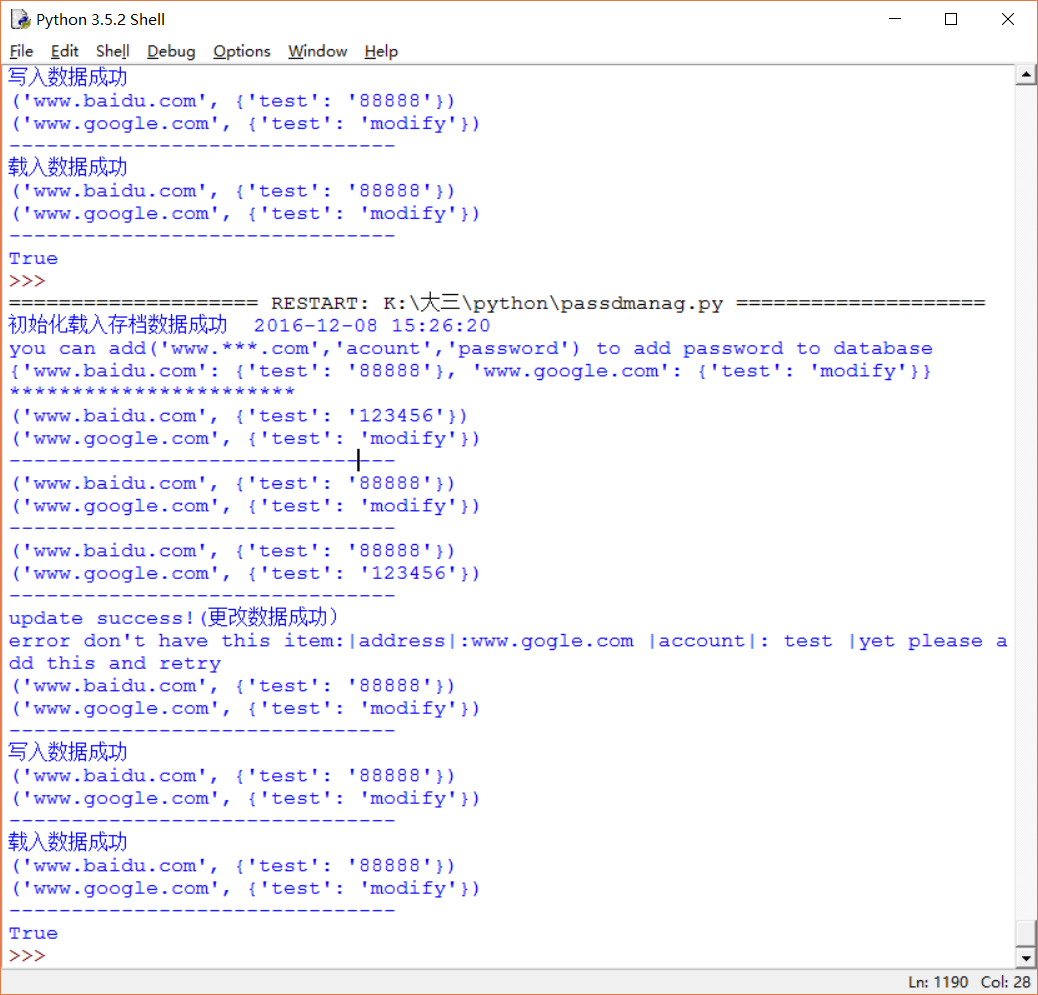
d.save()

d.showAll()

d.load()

d.showAll()

print(d.filecheck())#测试函数运行





**实验二：**

测试实例为随机的2000个列表数据，引入cProfil包来分析函数调用性能。

从时间角度来看，快速排序>归并排序>堆排序>选择排序>插入排序>冒泡排序,与数据结构课程分析较为吻合。

num : 2000

gene^------^------^------^---------------------------------------------

~~~~~~~~~~~~~~~~~~~~冒泡排序性能~~~~~~~~~~~~~~~~~~~~

5 function calls in 0.458 seconds

Ordered by: standard name

ncalls tottime percall cumtime percall filename:lineno(function)

1 0.000 0.000 0.458 0.458 <string>:1(<module>)

1 0.458 0.458 0.458 0.458 Python\_sort.py:15(bubbleSort)

1 0.000 0.000 0.458 0.458 {built-in method builtins.exec}

1 0.000 0.000 0.000 0.000 {built-in method builtins.len}

1 0.000 0.000 0.000 0.000 {method 'disable' of '\_lsprof.Profiler' objects}

num : 2000

gene^------^------^------^---------------------------------------------

~~~~~~~~~~~~~~~~~~~~选择排序性能~~~~~~~~~~~~~~~~~~~~

5 function calls in 0.231 seconds

Ordered by: standard name

ncalls tottime percall cumtime percall filename:lineno(function)

1 0.000 0.000 0.231 0.231 <string>:1(<module>)

1 0.231 0.231 0.231 0.231 Python\_sort.py:24(selectionSort)

1 0.000 0.000 0.231 0.231 {built-in method builtins.exec}

1 0.000 0.000 0.000 0.000 {built-in method builtins.len}

1 0.000 0.000 0.000 0.000 {method 'disable' of '\_lsprof.Profiler' objects}

num : 2000

gene^------^------^------^---------------------------------------------

~~~~~~~~~~~~~~~~~~~~插入排序性能~~~~~~~~~~~~~~~~~~~~

5 function calls in 0.247 seconds

Ordered by: standard name

ncalls tottime percall cumtime percall filename:lineno(function)

1 0.000 0.000 0.247 0.247 <string>:1(<module>)

1 0.247 0.247 0.247 0.247 Python\_sort.py:33(insertionSort)

1 0.000 0.000 0.247 0.247 {built-in method builtins.exec}

1 0.000 0.000 0.000 0.000 {built-in method builtins.len}

1 0.000 0.000 0.000 0.000 {method 'disable' of '\_lsprof.Profiler' objects}

num : 2000

gene^------^------^------^---------------------------------------------

~~~~~~~~~~~~~~~~~~~~归并排序性能~~~~~~~~~~~~~~~~~~~~

111169 function calls (103108 primitive calls) in 0.029 seconds

Ordered by: standard name

ncalls tottime percall cumtime percall filename:lineno(function)

1 0.000 0.000 0.029 0.029 <string>:1(<module>)

8062/1 0.021 0.000 0.029 0.029 Python\_sort.py:48(mergeSort)

1 0.000 0.000 0.029 0.029 {built-in method builtins.exec}

60155 0.003 0.000 0.003 0.000 {built-in method builtins.len}

20131 0.001 0.000 0.001 0.000 {method 'append' of 'list' objects}

1 0.000 0.000 0.000 0.000 {method 'disable' of '\_lsprof.Profiler' objects}

2687 0.000 0.000 0.000 0.000 {method 'extend' of 'list' objects}

20131 0.003 0.000 0.003 0.000 {method 'pop' of 'list' objects}

num : 2000

gene^------^------^------^---------------------------------------------

~~~~~~~~~~~~~~~~~~~~快速排序性能~~~~~~~~~~~~~~~~~~~~

43889 function calls (40091 primitive calls) in 0.011 seconds

Ordered by: standard name

ncalls tottime percall cumtime percall filename:lineno(function)

1 0.000 0.000 0.011 0.011 <string>:1(<module>)

3799/1 0.008 0.000 0.011 0.011 Python\_sort.py:66(quickSort)

1 0.000 0.000 0.011 0.011 {built-in method builtins.exec}

3799 0.000 0.000 0.000 0.000 {built-in method builtins.len}

34389 0.002 0.000 0.002 0.000 {method 'append' of 'list' objects}

1 0.000 0.000 0.000 0.000 {method 'disable' of '\_lsprof.Profiler' objects}

1899 0.000 0.000 0.000 0.000 {method 'pop' of 'list' objects}

num : 2000

gene^------^------^------^---------------------------------------------

~~~~~~~~~~~~~~~~~~~~堆排序性能~~~~~~~~~~~~~~~~~~~~

21009 function calls (3006 primitive calls) in 0.037 seconds

Ordered by: standard name

ncalls tottime percall cumtime percall filename:lineno(function)

1 0.000 0.000 0.036 0.036 <string>:1(<module>)

1 0.001 0.001 0.036 0.036 Python\_sort.py:105(heapSort)

21002/2999 0.035 0.000 0.035 0.000 Python\_sort.py:82(maxHeapify)

1 0.000 0.000 0.005 0.005 Python\_sort.py:96(buildMaxHeap)

1 0.000 0.000 0.037 0.037 {built-in method builtins.exec}

2 0.000 0.000 0.000 0.000 {built-in method builtins.len}

1 0.000 0.000 0.000 0.000 {method 'disable' of '\_lsprof.Profiler' objects}

import random

import cProfile

unsortedList=[]

# generate an unsorted list

def generateUnsortedList(num):

unsortedList.clear()

for i in range(0,num):

unsortedList.append(random.randint(0,100))

#print(unsortedList)

print('num :',num)

print('gene^------^------^------^---------------------------------------------')

#####################################################################代码

#一， 冒泡排序

def bubbleSort(unsortedList):

list\_length=len(unsortedList)

for i in range(0,list\_length-1):

for j in range(0,list\_length-i-1):

if unsortedList[j]>unsortedList[j+1]:

unsortedList[j],unsortedList[j+1]=unsortedList[j+1],unsortedList[j]

return unsortedList

#二，选择排序

def selectionSort(unsortedList):

list\_length=len(unsortedList)

for i in range(0,list\_length-1):

for j in range(i+1,list\_length):

if unsortedList[i]>unsortedList[j]:

unsortedList[i],unsortedList[j]=unsortedList[j],unsortedList[i]

return unsortedList

#三、插入排序

def insertionSort(unsortedList):

list\_length=len(unsortedList)

if list\_length<2:

return unsortedList

for i in range(1,list\_length):

key=unsortedList[i]

j=i-1

while j>=0 and key<unsortedList[j]:

unsortedList[j+1]=unsortedList[j]

j=j-1

unsortedList[j+1]=key

return unsortedList

#四，归并排序

def mergeSort(unsortedList):

if len(unsortedList)<2:

return unsortedList

sortedList=[]

left=mergeSort(unsortedList[:int(len(unsortedList)/2)])

right=mergeSort(unsortedList[int(len(unsortedList)/2):])

while len(left)>0 and len(right)>0:

if left[0]<right[0]:

sortedList.append(left.pop(0))

else:

sortedList.append(right.pop(0))

if len(left)>0:

sortedList.extend(mergeSort(left))

else:

sortedList.extend(mergeSort(right))

return sortedList

#五，快速排序

def quickSort(unsortedList):

if len(unsortedList)<2:

return unsortedList

less=[]

greater=[]

middle=unsortedList.pop(0)

for item in unsortedList:

if item<middle:

less.append(item)

else:

greater.append(item)

return quickSort(less)+[middle]+quickSort(greater)

#六，堆排序

def maxHeapify(L,heap\_size,i):

leftChildIndex=2\*i+1

rightChildIndex=2\*i+2

# print 'leftChildIndex=',leftChildIndex

# print 'rightChildIndex=',rightChildIndex

largest=i

if leftChildIndex<heap\_size and L[int(leftChildIndex)]>L[int(largest)]:

largest=leftChildIndex

if rightChildIndex<heap\_size and L[int(rightChildIndex)]>L[int(largest)]:

largest=rightChildIndex

if largest!=i:

L[int(i)],L[int(largest)]=L[int(largest)],L[int(i)]

maxHeapify(L,heap\_size,largest)

def buildMaxHeap(L):

heap\_size=len(L)

if heap\_size>1:

start=heap\_size/2-1

# print 'start=',start

while start>=0:

maxHeapify(L,heap\_size,start)

start=start-1

def heapSort(L):

heap\_size=len(L)

buildMaxHeap(L)

i=heap\_size-1

while i>0:

L[0],L[i]=L[i],L[0]

heap\_size=heap\_size-1

i=i-1

maxHeapify(L,heap\_size,0)

return L

######################################测试module

if \_\_name\_\_=='\_\_main\_\_':

generateUnsortedList(2000)

print('~~~~~~~~~~~~~~~~~~~~冒泡排序性能~~~~~~~~~~~~~~~~~~~~')

cProfile.run('bubbleSort(unsortedList)')

generateUnsortedList(2000)

print('~~~~~~~~~~~~~~~~~~~~选择排序性能~~~~~~~~~~~~~~~~~~~~')

cProfile.run('selectionSort(unsortedList)')

generateUnsortedList(2000)

print('~~~~~~~~~~~~~~~~~~~~插入排序性能~~~~~~~~~~~~~~~~~~~~')

cProfile.run('insertionSort(unsortedList)')

generateUnsortedList(2000)

print('~~~~~~~~~~~~~~~~~~~~归并排序性能~~~~~~~~~~~~~~~~~~~~')

cProfile.run('mergeSort(unsortedList)')

generateUnsortedList(2000)

print('~~~~~~~~~~~~~~~~~~~~快速排序性能~~~~~~~~~~~~~~~~~~~~')

cProfile.run('quickSort(unsortedList)')

generateUnsortedList(2000)

print('~~~~~~~~~~~~~~~~~~~~堆排序性能~~~~~~~~~~~~~~~~~~~~')

cProfile.run('heapSort(unsortedList)')