# 資料結構：作業三

108701009 應數一 許書睿

Python 3.7:

Part A : 定義data structure和function

1. Binary search tree:

<https://www.tutorialspoint.com/python_data_structure/python_binary_search_tree.htm>

class Node:

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

self.left = None

self.right = None

self.data = data

# Insert method to create nodes

def insert(self, data):

if self.data:

if data < self.data:

if self.left is None:

self.left = Node(data)

else:

self.left.insert(data)

elif data > self.data:

if self.right is None:

self.right = Node(data)

else:

self.right.insert(data)

else:

self.data = data

# findval method to compare the value with nodes

def findval(self, lkpval):

if lkpval < self.data:

if self.left is None:

return str(lkpval)+" Not Found"

return self.left.findval(lkpval)

elif lkpval > self.data:

if self.right is None:

return str(lkpval)+" Not Found"

return self.right.findval(lkpval)

else:

print(str(self.data) + ' is found')

# Print the tree

def PrintTree(self):

if self.left:

self.left.PrintTree()

print( self.data),

if self.right:

self.right.PrintTree()

1. Skip list::

<https://www.geeksforgeeks.org/skip-list-set-2-insertion/>

import random

class Node2(object):

'''

Class to implement node

'''

def \_\_init\_\_(self, key, level):

self.key = key

# list to hold references to node of different level

self.forward = [None]\*(level+1)

class SkipList(object):

'''

Class for Skip list

'''

def \_\_init\_\_(self, max\_lvl, P):

# Maximum level for this skip list

self.MAXLVL = max\_lvl

# P is the fraction of the nodes with level

# i references also having level i+1 references

self.P = P

# create header node and initialize key to -1

self.header = self.createNode(self.MAXLVL, -1)

# current level of skip list

self.level = 0

# create new node

def createNode(self, lvl, key):

n = Node2(key, lvl)

return n

# create random level for node

def randomLevel(self):

lvl = 0

while random.random()<self.P and lvl<self.MAXLVL:lvl += 1

return lvl

# insert given key in skip list

def insertElement(self, key):

# create update array and initialize it

update = [None]\*(self.MAXLVL+1)

current = self.header

'''

start from highest level of skip list

move the current reference forward while key

is greater than key of node next to current

Otherwise inserted current in update and

move one level down and continue search

'''

for i in range(self.level, -1, -1):

while current.forward[i] and current.forward[i].key < key:

current = current.forward[i]

update[i] = current

'''

reached level 0 and forward reference to

right, which is desired position to

insert key.

'''

current = current.forward[0]

'''

if current is NULL that means we have reached

to end of the level or current's key is not equal

to key to insert that means we have to insert

node between update[0] and current node

'''

if current == None or current.key != key:

# Generate a random level for node

rlevel = self.randomLevel()

'''

If random level is greater than list's current

level (node with highest level inserted in

list so far), initialize update value with reference

to header for further use

'''

if rlevel > self.level:

for i in range(self.level+1, rlevel+1):

update[i] = self.header

self.level = rlevel

# create new node with random level generated

n = self.createNode(rlevel, key)

# insert node by rearranging references

for i in range(rlevel+1):

n.forward[i] = update[i].forward[i]

update[i].forward[i] = n

print("Successfully inserted key {}".format(key))

# Display skip list level wise

def displayList(self):

print("\n\*\*\*\*\*Skip List\*\*\*\*\*\*")

head = self.header

for lvl in range(self.level+1):

print("Level {}: ".format(lvl), end=" ")

node = head.forward[lvl]

while(node != None):

print(node.key, end=" ")

node = node.forward[lvl]

print("")

def searchElement(self, key):

current = self.header

'''

start from highest level of skip list

move the current reference forward while key

is greater than key of node next to current

Otherwise inserted current in update and

move one level down and continue search

'''

for i in range(self.level, -1, -1):

while(current.forward[i] and\

current.forward[i].key < key):

current = current.forward[i]

# reached level 0 and advance reference to

# right, which is prssibly our desired node

current = current.forward[0]

# If current node have key equal to

# search key, we have found our target node

if current and current.key == key:

print("Found key ", key)

3.sorted array:

<https://www.geeksforgeeks.org/search-insert-and-delete-in-a-sorted-array/>

def binarySearch(arr, low, high, key):

if (high < low):

return -1

# low + (high - low)/2

mid = (low + high)/2

if (key == arr[int(mid)]):

return mid

if (key > arr[int(mid)]):

return binarySearch(arr,

(mid + 1), high, key)

return (binarySearch(arr, low,

(mid -1), key))

# This code is contributed by

# Smitha Dinesh Semwal

def insertSorted(arr, n, key, capacity):

# Cannot insert more elements if n is

# already more than or equal to capcity

if (n >= capacity):

return n

i = n - 1

while i >= 0 and arr[i] > key:

arr[i + 1] = arr[i]

i -= 1

arr[i + 1] = key

return (n + 1)

生成隨機數

import numpy as np

def random\_array(k):

return np.random.randint(1,2\*\*k,size=2\*\*k)

以及 時間模塊

import time

Part B : 測試方式

（以bst為例）

t\_insert = []

t\_search = []

for k in range(10,31):

#insert part

t\_insert\_ =[]

for times in range(10):

insert\_array = random\_array(k)

'這個binary tree無法創建空的資料結構，所以先包含了一個資料，影響應該不大'

bst = Node1(np.random.randint(1,2\*\*30))

t = time.perf\_counter()

for i in insert\_array:

bst.insert(i)

t = time.perf\_counter() - t

t\_insert\_.append(t)

print('\rk=', k,' times:',times+1,' time:',time.strftime("%Y-%m-%d %H:%M:%S", time.localtime()),t\_insert\_,end=' ')

t\_insert.append(np.mean(t\_insert\_))

print('\ninsert: k=',k, t\_insert)

#search part

t\_search\_=[]

for times in range(10):

search\_array = random\_array(17) #這邊要十萬筆資料（2\*\*16.6)

t = time.perf\_counter()

for i in search\_array:

bst.findval(i)

t = time.perf\_counter() - t

t\_search\_.append(t)

print('\rk=', k,' times:',times+1,' time:',time.strftime("%Y-%m-%d %H:%M:%S", time.localtime()),t\_search\_,end='')

t\_search.append(np.mean(t\_search\_))

print('\nsearch: k=',k, t\_search)

Note:

這邊的 output 長這樣：

…

k= 12 times: 10 time: 2020-06-21 17:30:41 [0.021991199999320088, 0.019774512998992577, 0.021150496000700514, 0.02051666999977897, 0.019767460000366555, 0.019976628000222263, 0.019516948999807937, 0.018582016000436852, 0.022263453000050504, 0.021592855999188032]

insert: k= 12 [0.004908217600132048, 0.009805989599772146, 0.02051322409988643]

k= 12 times: 10 time: 2020-06-21 17:30:48 [0.6767590710005607, 0.6577993680002692, 0.6143646970012924, 0.6445872789990972, 0.6567319780006073, 0.6442201270001533, 0.6737532049992296, 0.6749322819996451, 0.6696228540004086, 0.6553539309988992]

search: k= 12 [0.5356921934004276, 0.7324735537002198, 0.6568124792000163]

…

觀察：如果 time後面的時間距現在時刻超過一個小時，則停止。

這樣做的原因是在如果不幸dead kernel 還能看到跑完的數據

然後令

bst\_insert = t\_insert

bst\_search=t\_search

用於繪圖

Part C : 估計時間

估計方法：假設等比或等差，看（公）差和（公）比，再做估計

1.檢查比（差）:

def checkway(t\_list):

print('比值：')

r=[]

for i in range(len(t\_list)-1):

e = t\_list[i+1]/t\_list[i]

print(e)

r.append(e)

print('平均比值',np.mean(r))

print('差：')

d=[]

for i in range(len(t\_list)-1):

e = t\_list[i+1]-t\_list[i]

print(e)

d.append(e)

print('平均差',np.mean(d))

2.選定方法和平均值作代表，然後套用:

def estim(t\_list,way,r):

if way=='r':

while len(t\_list)<21:

t\_list.append(t\_list[-1]\*r)

print(t\_list)

elif way=='d':

while len(t\_list)<21:

t\_list.append(t\_list[-1]+r)

print(t\_list)

else:

print('error')

舉例：bst\_insert

checkway(bst\_insert)

output:

比值：

1.9978718138960123

2.091907592922909

2.6065622565915496

2.0679689949827034

5.740032557704272

1.4386147285474231

2.131662197492198

2.1090824759664875

2.2553035119615323

2.058341139028241

2.1798335296139353

2.0952536942050206

2.138322155885444

2.1662345851860385

平均比值 2.36264223099884

差：

0.0048977719996400985

0.010707234500114283

0.0329557715998817

0.05710322960021585

0.524115947899736

0.27838358080025505

1.0332887875996675

2.1586743686008045

5.153064739499315

9.798227728500205

22.483293992100336

45.496423761699226

99.07505853440053

217.04920578289986

平均差 28.796814373692843

這邊用公比，比值取平均值

estim(bst\_insert,'r',2.36264223099884)

output:

[0.004908217600132048, 0.009805989599772146, 0.02051322409988643, 0.05346899569976813, 0.11057222529998398, 0.63468817319972, 0.9130717539999751, 1.9463605415996426, 4.105034910200447, 9.258099649699762, 19.056327378199967, 41.5396213703003, 87.03604513199953, 186.11110366640006, 403.1603094492999, 952.5235729674768, 2250.472419514866, 5317.061178043961, 12562.313284051106, 29680.25188393687, 70123.81652767214]

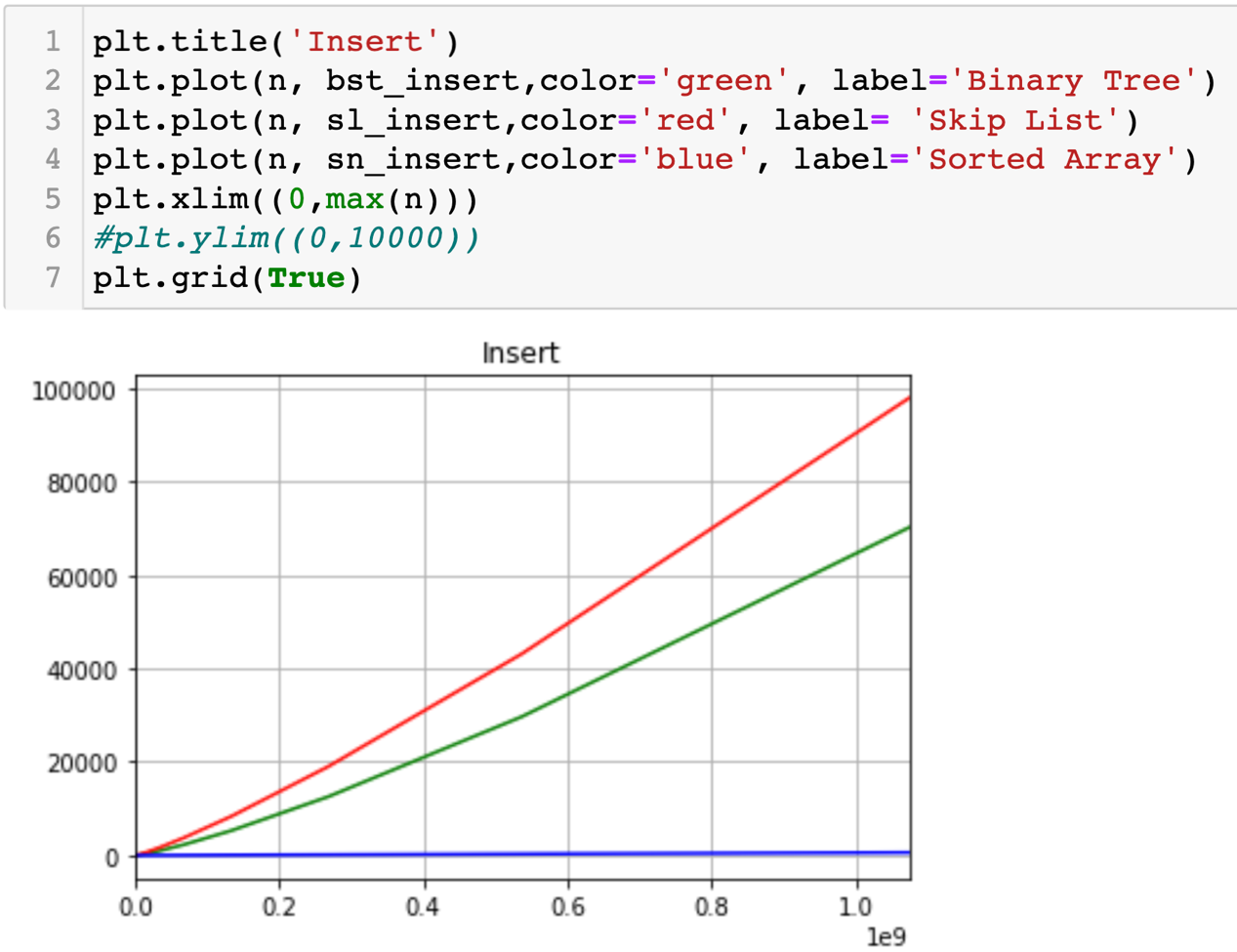
Part D: 結果

紅色：binary tree

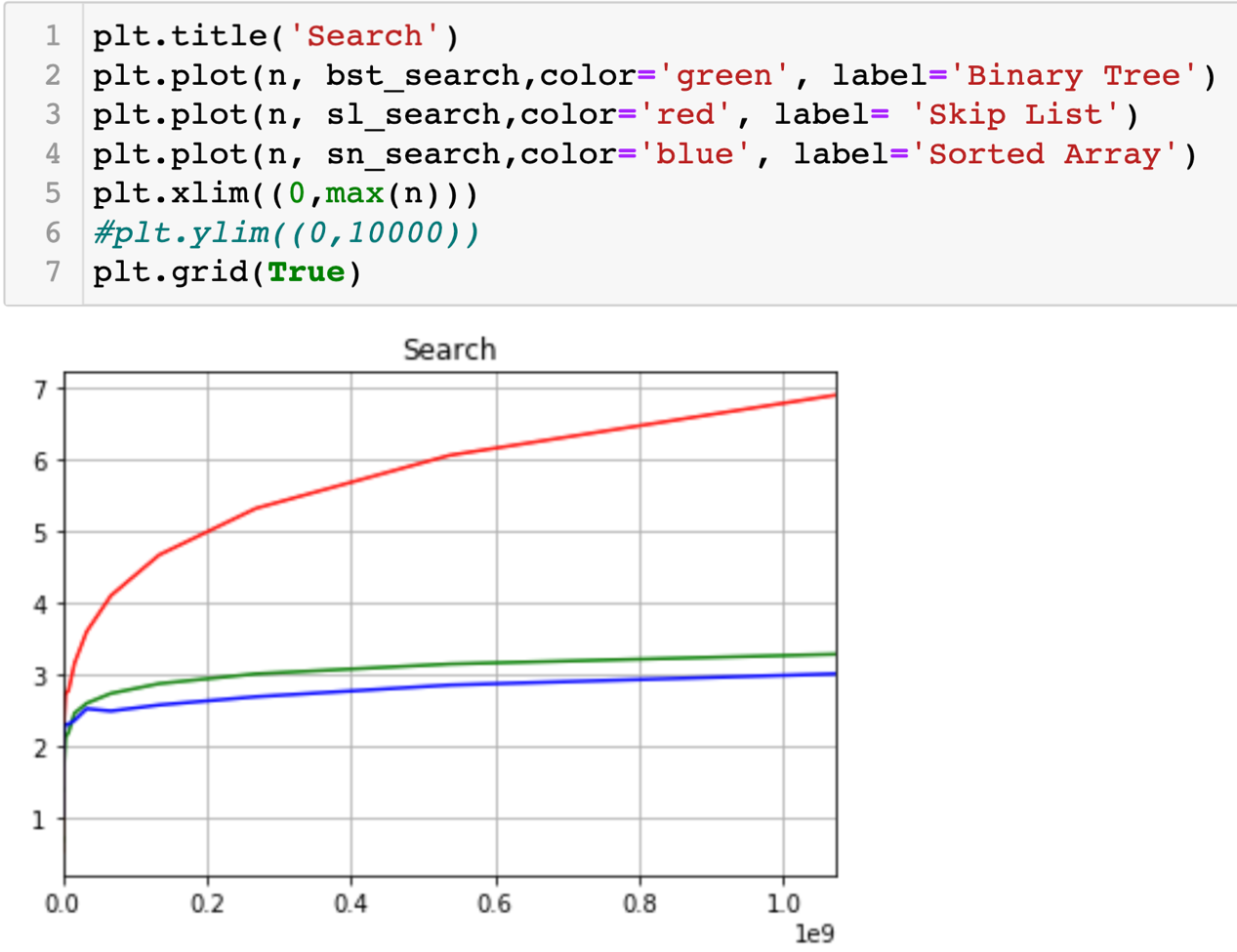
綠色：skip list

藍色：sorted array

1. Insert:



2.Search:



Part E: 其他

作業系統：macOS Ctalina 10.15.3，CPU: i5-8259U，RAM:8GB

Code在：<https://nbviewer.jupyter.org/github/ivan1003hsu/python_work/blob/master/for_DS/HW_3.ipynb>