HW3

P76074402 張圜華

實作環境: python

實作部分說明:

p, p\_, count = read\_file(filename) ->這一行是給 graph 1~6

#p, p\_, count = read\_outfile(filename2) ->這一行是給 direct

#p, p\_, count = read\_outfile\_bi(filename2) ->這一行是給 bi - direct

Hit

1. Hub與Authority

Hub和Authority是HITS算法最基本的兩個定義。

Authority 是指與某個領域或者某個話題相關的高質量網頁，比如搜索引擎領域，Google和百度首頁即該領域的高質量網頁，比如視頻領域，優酷和土豆首頁即該領域的高質量網頁。

  Hub 是指的是包含了很多指向高質量“Authority”頁面鏈接的網頁，比如

hao123首頁可以認為是一個典型的高質量“Hub”網頁。

HITS算法的目的即是通過一定的技術手段，在海量網頁中找到與用戶查詢主題相關的高質量“Authority”頁面和“Hub”頁面，尤其是“Authority”頁面，因為這些頁面代表了能夠滿足用戶查詢的高質量內容，搜索引擎以此作為搜索結果返回給用戶。

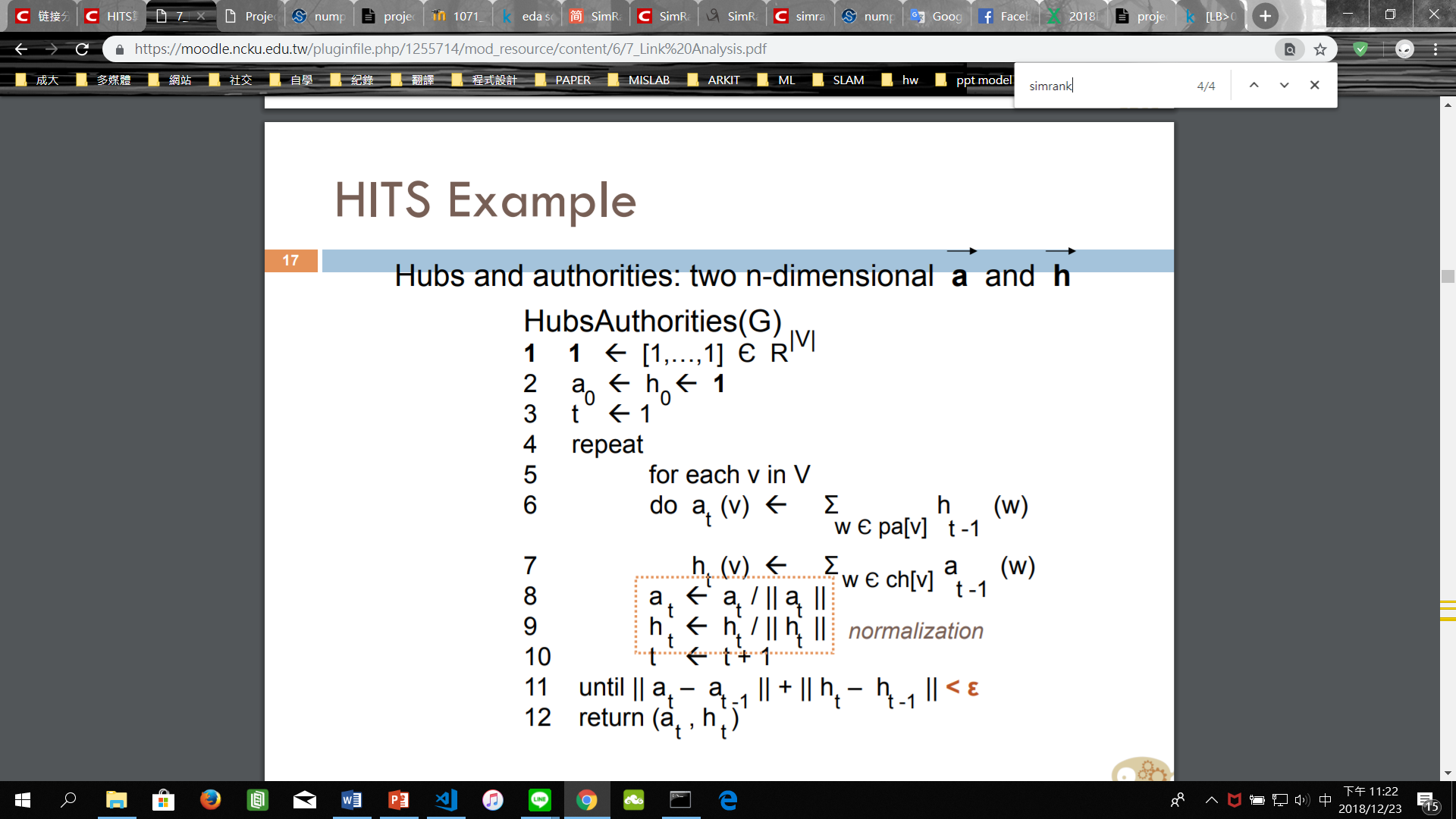
2. 基本思想

     一個好的“Authority”頁面會被很多好的“Hub”頁面指向；

     一個好的“Hub”頁面會指向很多好的“Authority”頁面；

實作細節:

依照下列演算法實作:



def hit(a, h):

a\_ = np.dot(a.T,h)

h\_ = np.dot(a,h)

a\_ = a\_/np.sum(a\_)

h\_ = h\_/np.sum(h\_)

e = 1

while (e > 0.00001):

past\_a = a\_

past\_h = h\_

a\_ = np.dot(a.T,past\_h)

h\_ = np.dot(a,past\_a)

a\_ = a\_/np.sum(a\_) #normalize

h\_ = h\_/np.sum(h\_) #normalize

e = np.linalg.norm(a\_-past\_a)+np.linalg.norm(h\_-past\_h)

return a\_, h\_

a 是矩陣，表示網頁之間的關係：aij=1表示網頁i指向網頁j，否則為0

a\_ 是向量，存所有頁面的authority值，其中第i個分量表示網頁i的authority值

h\_ 是向量，存所有頁面的hub值，其中第i個分量表示網頁i的hub值

e 是指 誤差

公式:

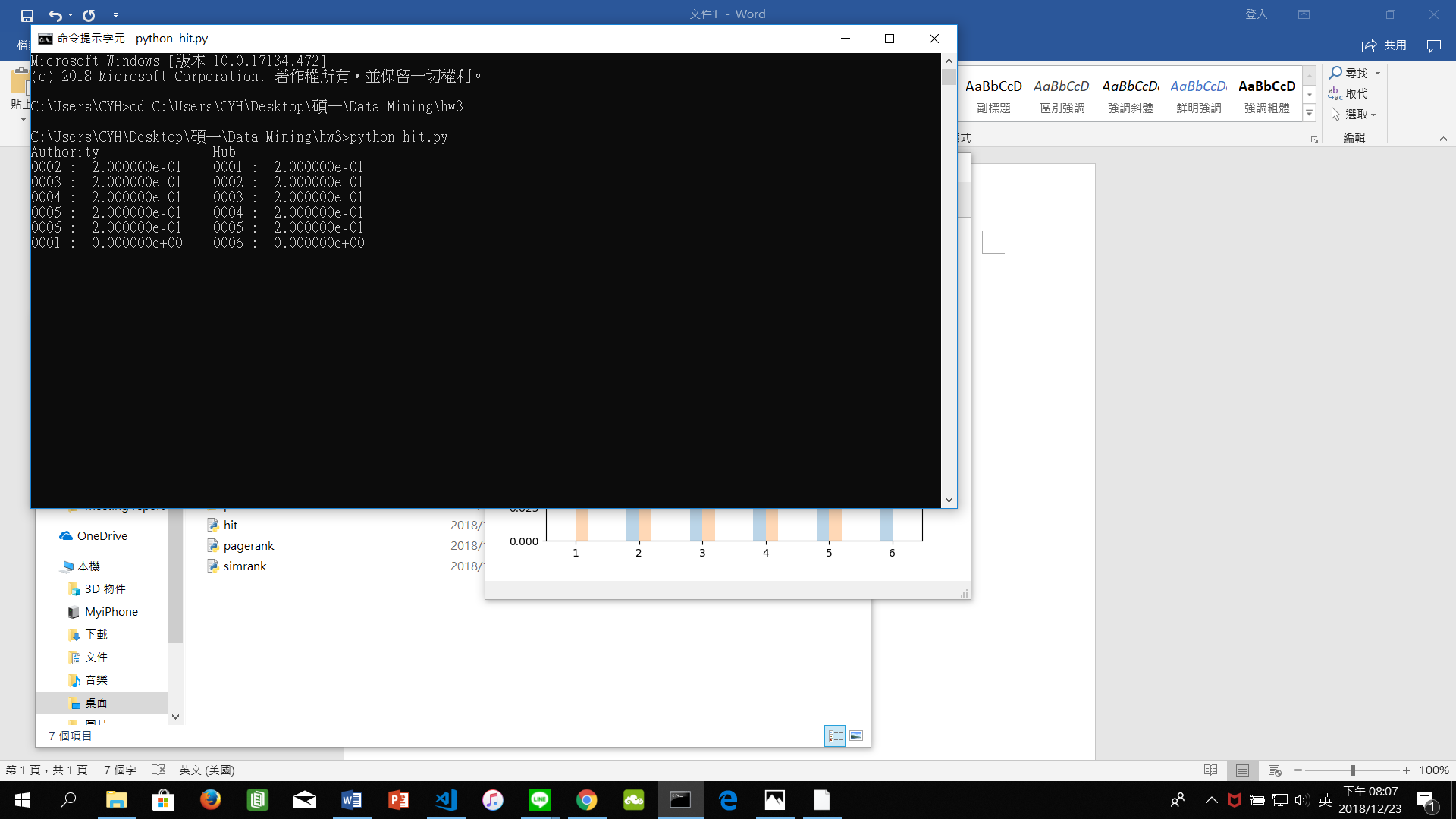
a\_(t) = np.dot(a.T, h(t-1))

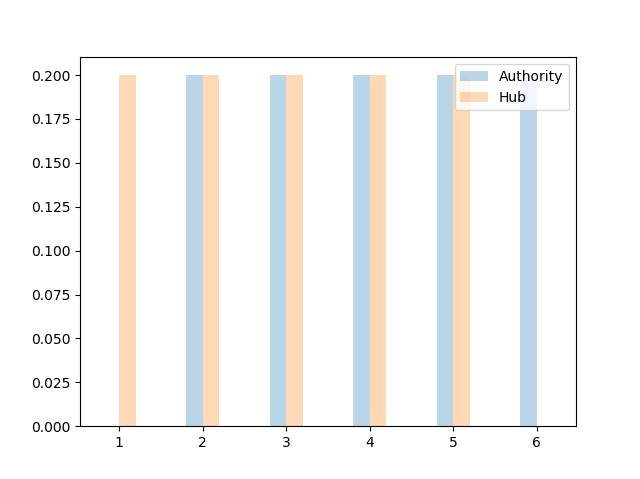
h\_(t) = np.dot(a, a\_(t-1))

直到 ||a\_(t)-a\_(t-1)||+ ||h\_(t)-h\_(t-1)|| < e 才停止

實作結果分析:

1. graph 1





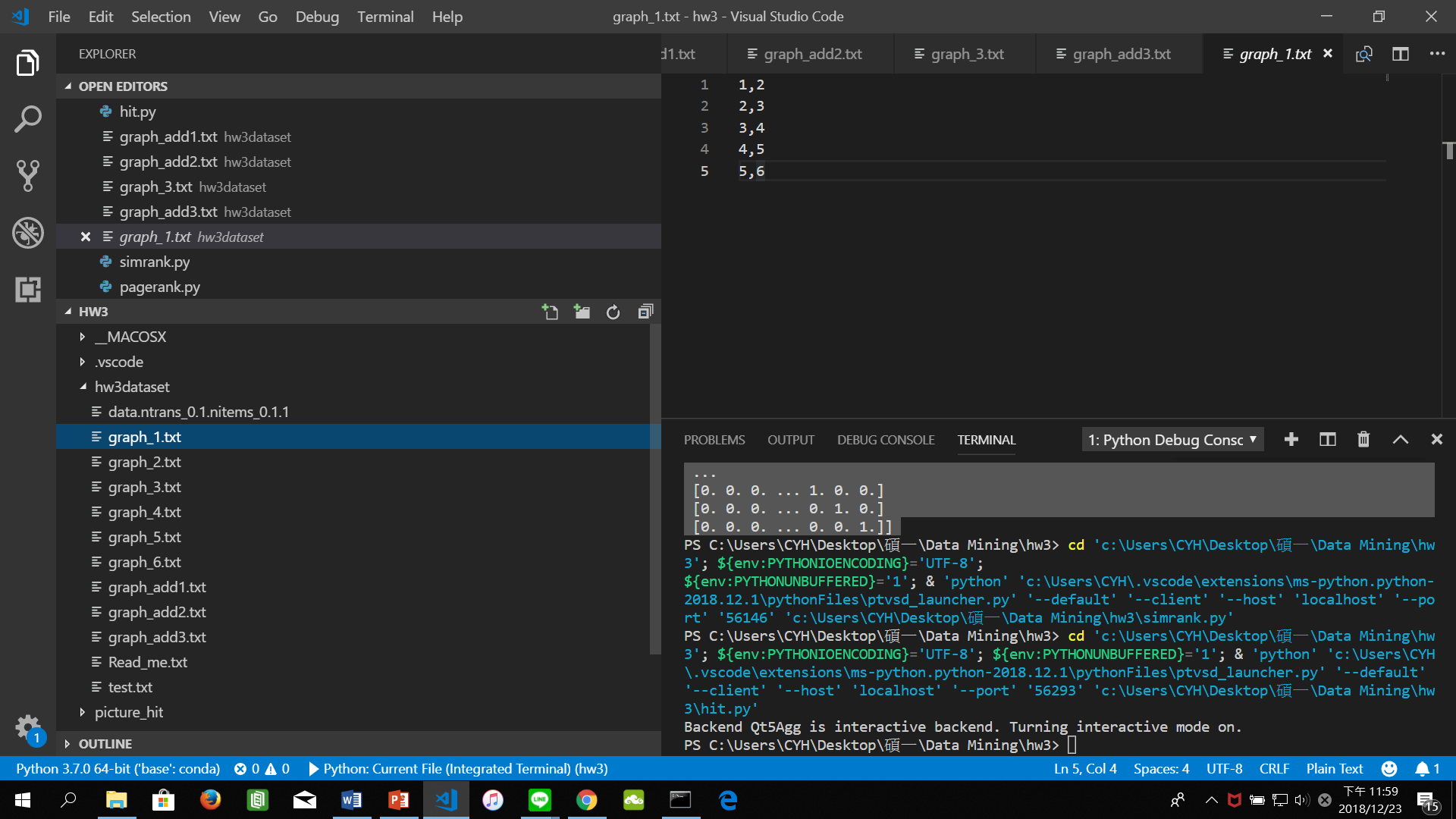
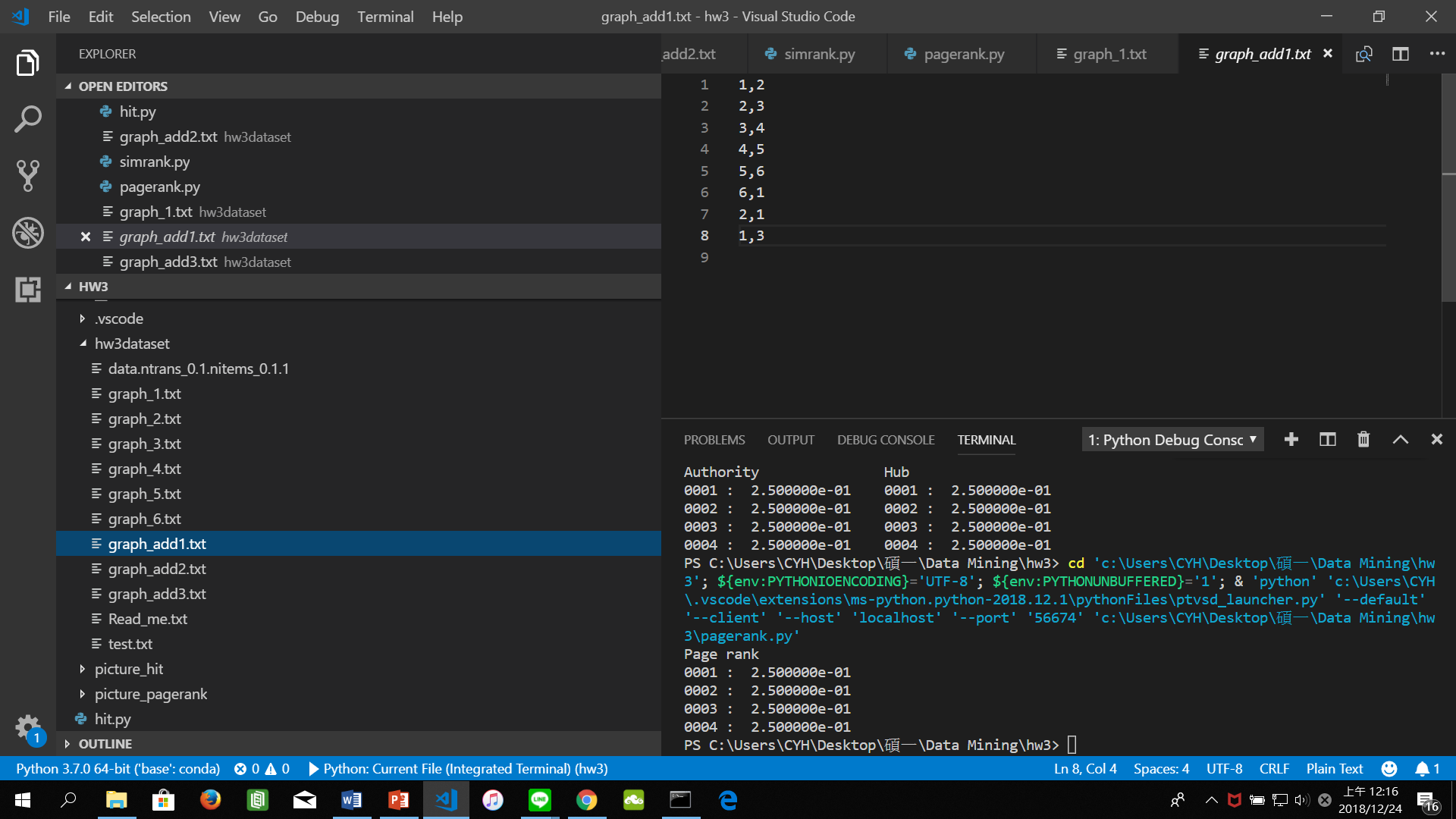
authority 大小順序: 2 3 4 5 6 1

hub 大小順序: 1 2 3 4 5 6

authority : 2 3 4 5 6 相同是因為都有一個node指向他們，而 1 沒有

hub : 1 2 3 4 5 相同是因為他們都有指向一個node，而 6 沒有

**修改graph1: graph\_add1.txt**

 -> 

Authority Hub

0003 : 4.450415e-01 0002 : 4.450415e-01

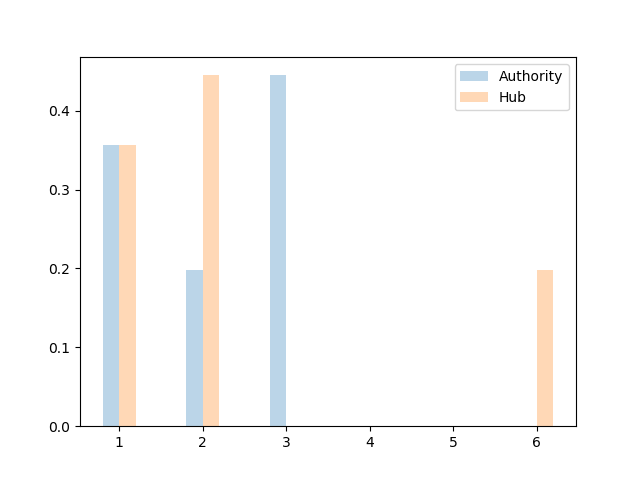
0001 : 3.568971e-01 0001 : 3.568971e-01

0002 : 1.980614e-01 0006 : 1.980614e-01

0004 : 1.397788e-08 0003 : 1.397788e-08

0005 : 1.397788e-08 0004 : 1.397788e-08

0006 : 1.397788e-08 0005 : 1.397788e-08

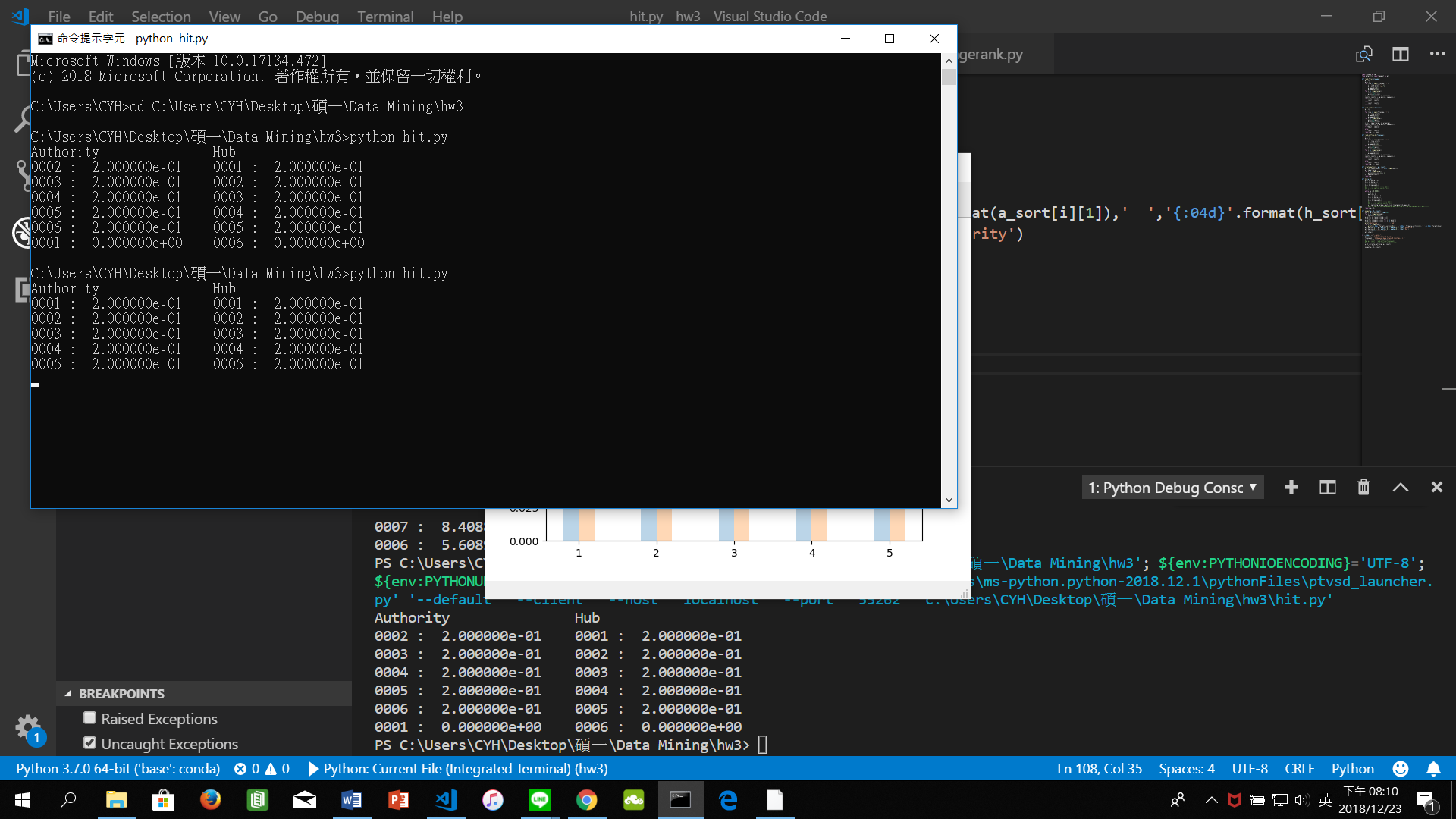


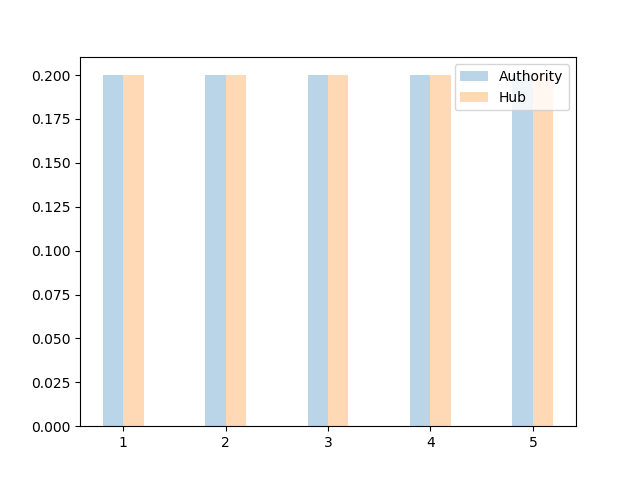
原本 1 的Authority:0 Hub:0.2

後來 1 的Authority: 3.568971e-01 Hub: 3.568971e-01

都有增加

1. graph 2





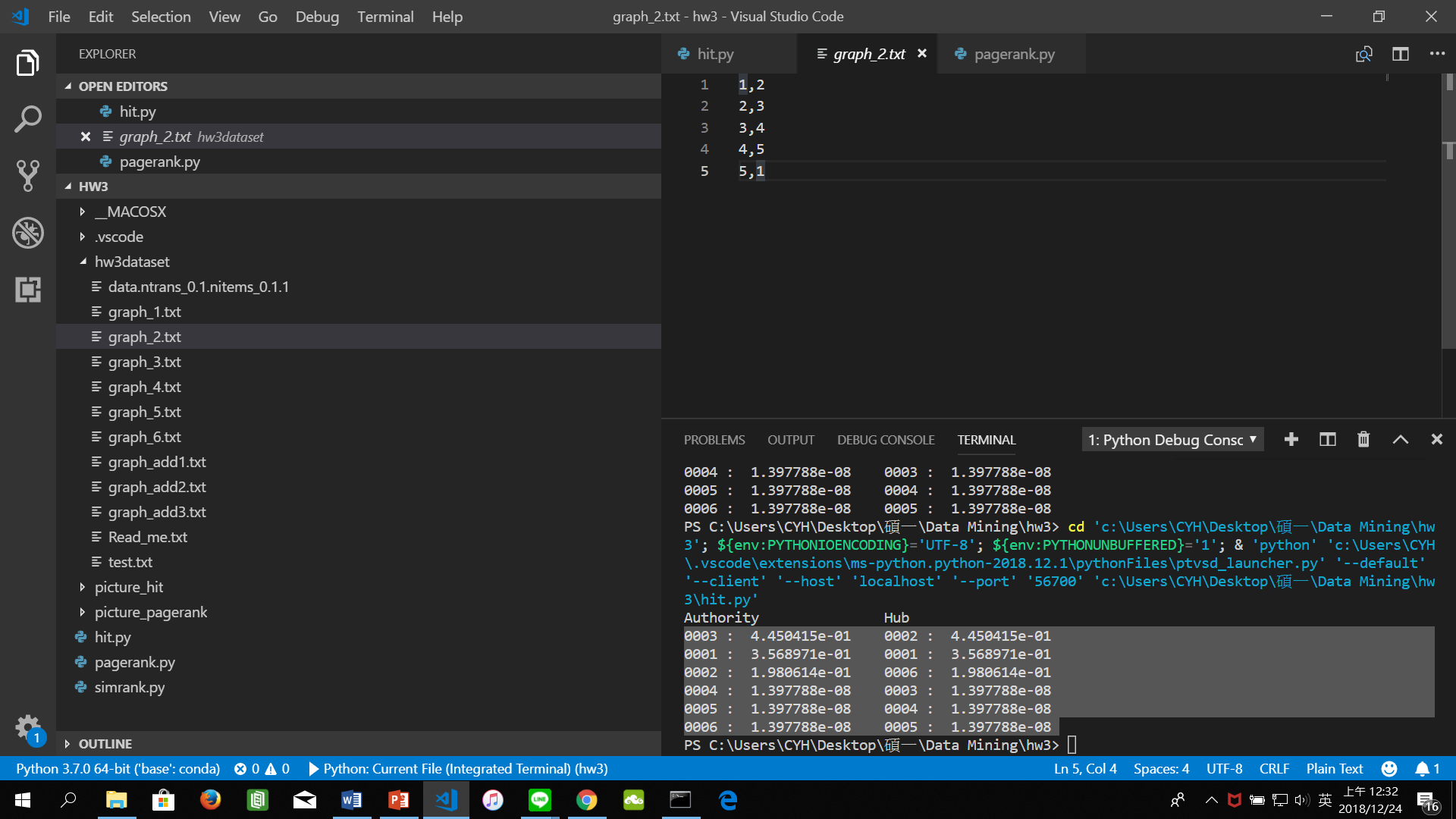
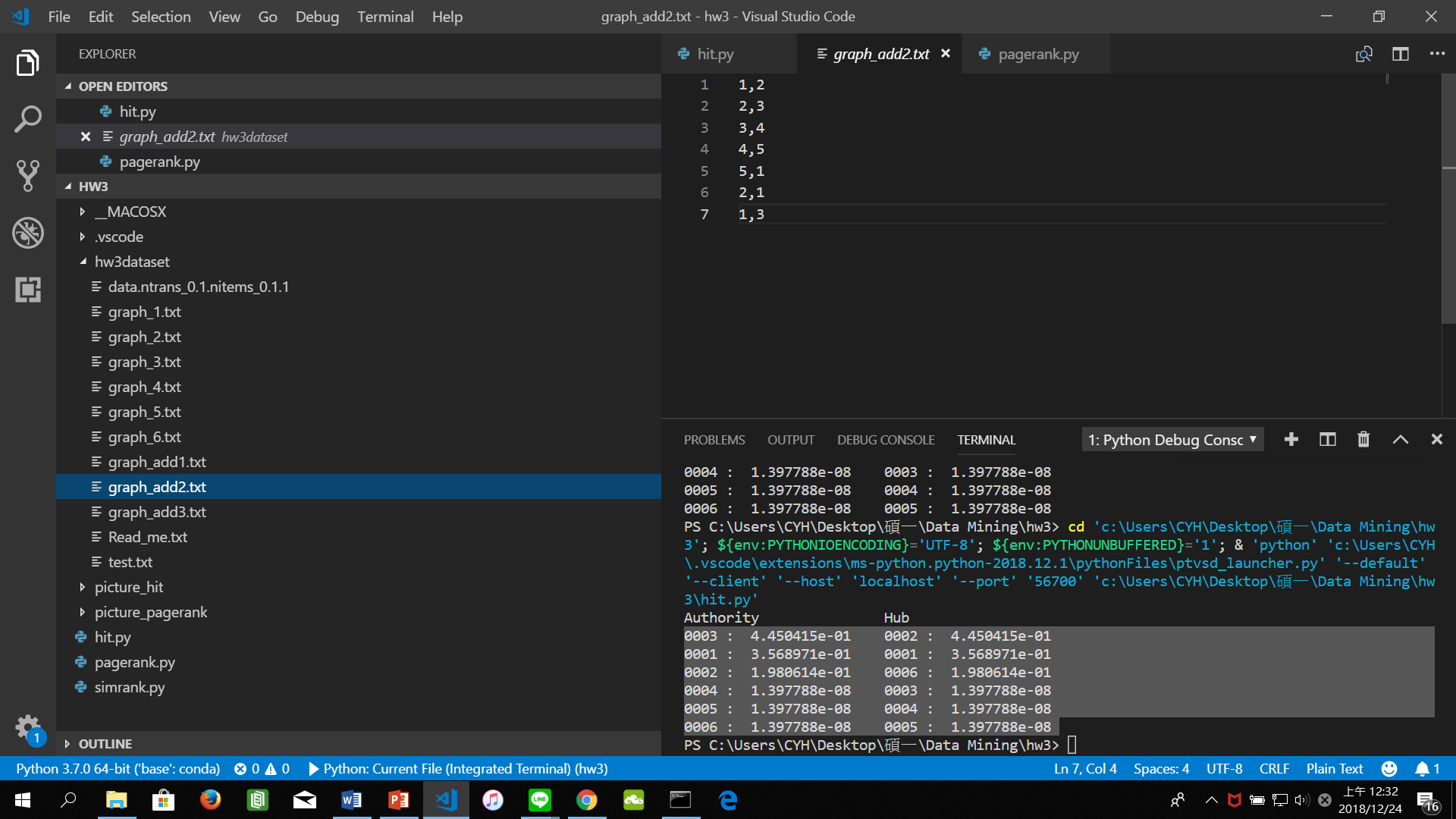
authority 大小順序: 1 2 3 4 5

hub 大小順序: 1 2 3 4 5

authority : 1 2 3 4 5 相同是因為都有一個node指向他們

hub : 1 2 3 4 5 相同是因為他們都有指向一個node

**修改graph2: graph\_add2.txt**

 **->** 

Authority Hub

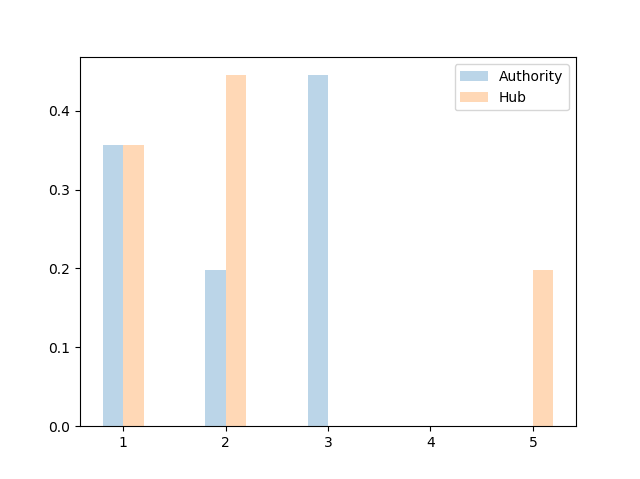
0003 : 4.450415e-01 0002 : 4.450415e-01

0001 : 3.568971e-01 0001 : 3.568971e-01

0002 : 1.980614e-01 0005 : 1.980614e-01

0004 : 1.397788e-08 0003 : 1.397788e-08

0005 : 1.397788e-08 0004 : 1.397788e-08

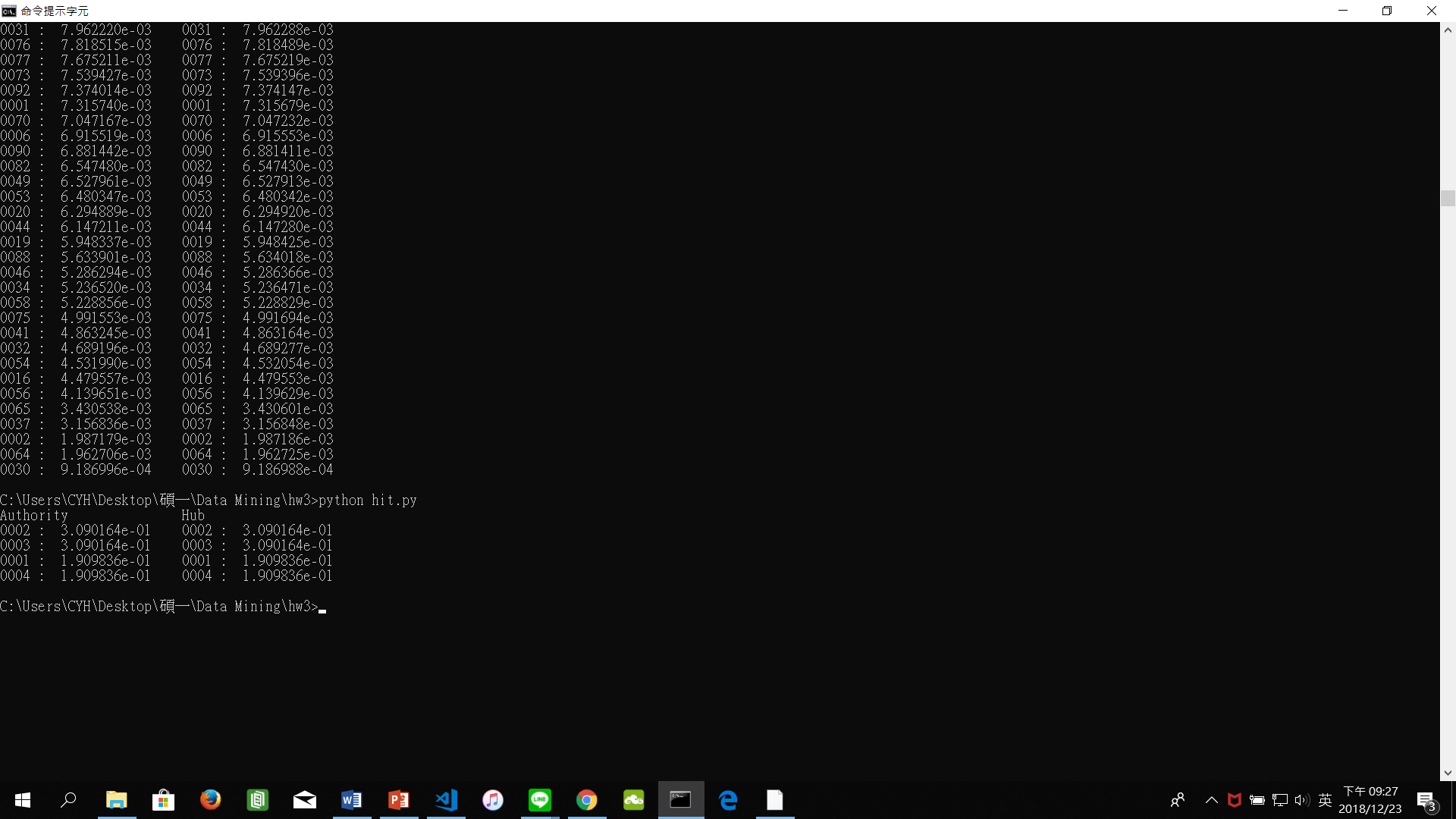


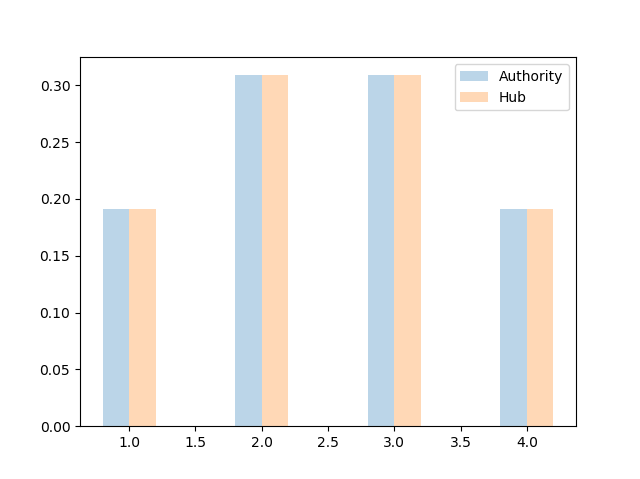
原本 1 的Authority:0.2 Hub:0.2

後來 1 的Authority: 3.568971e-01 Hub: 3.568971e-01

都有增加

1. graph 3





authority 大小順序: 2 3 1 4

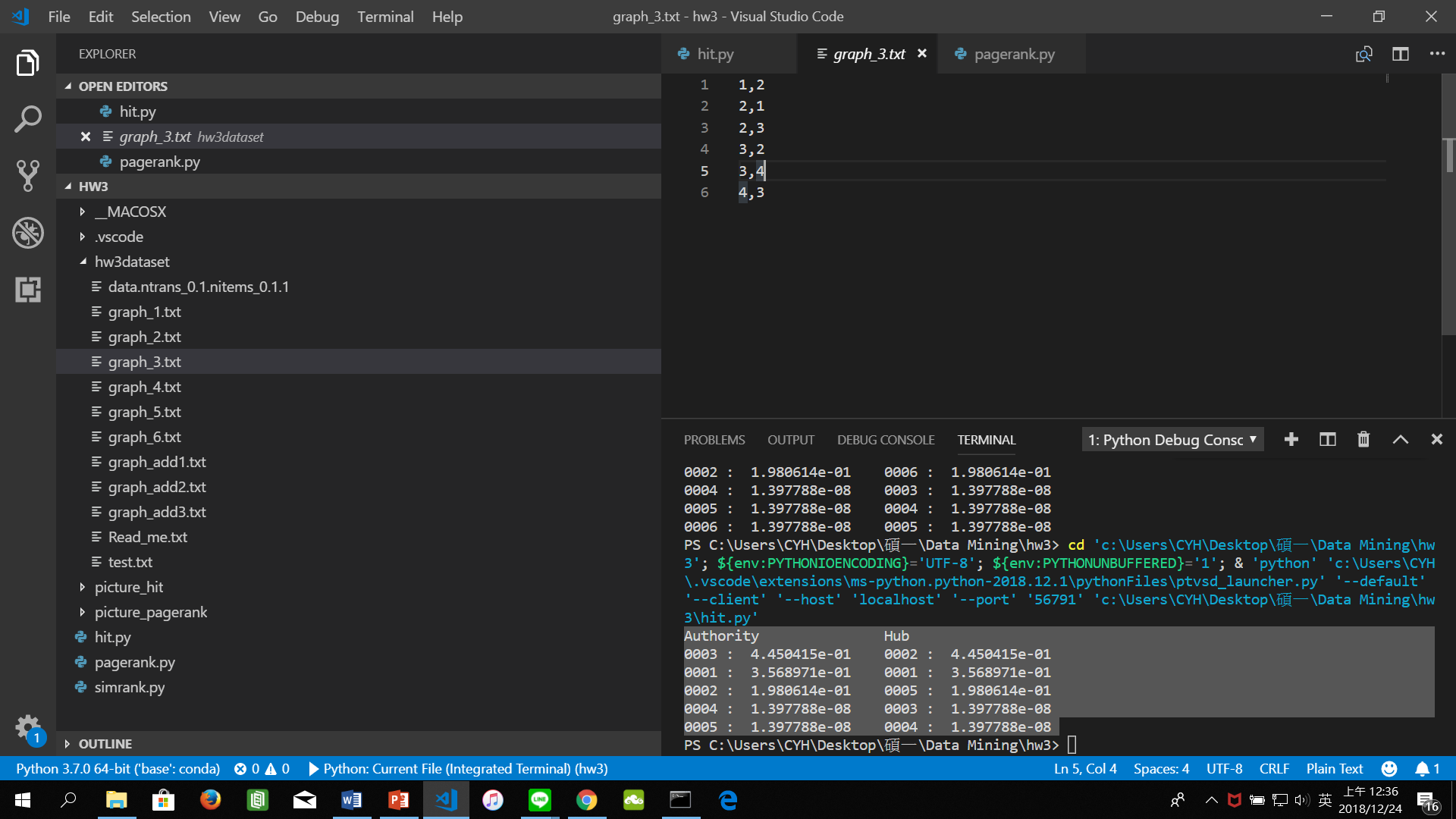
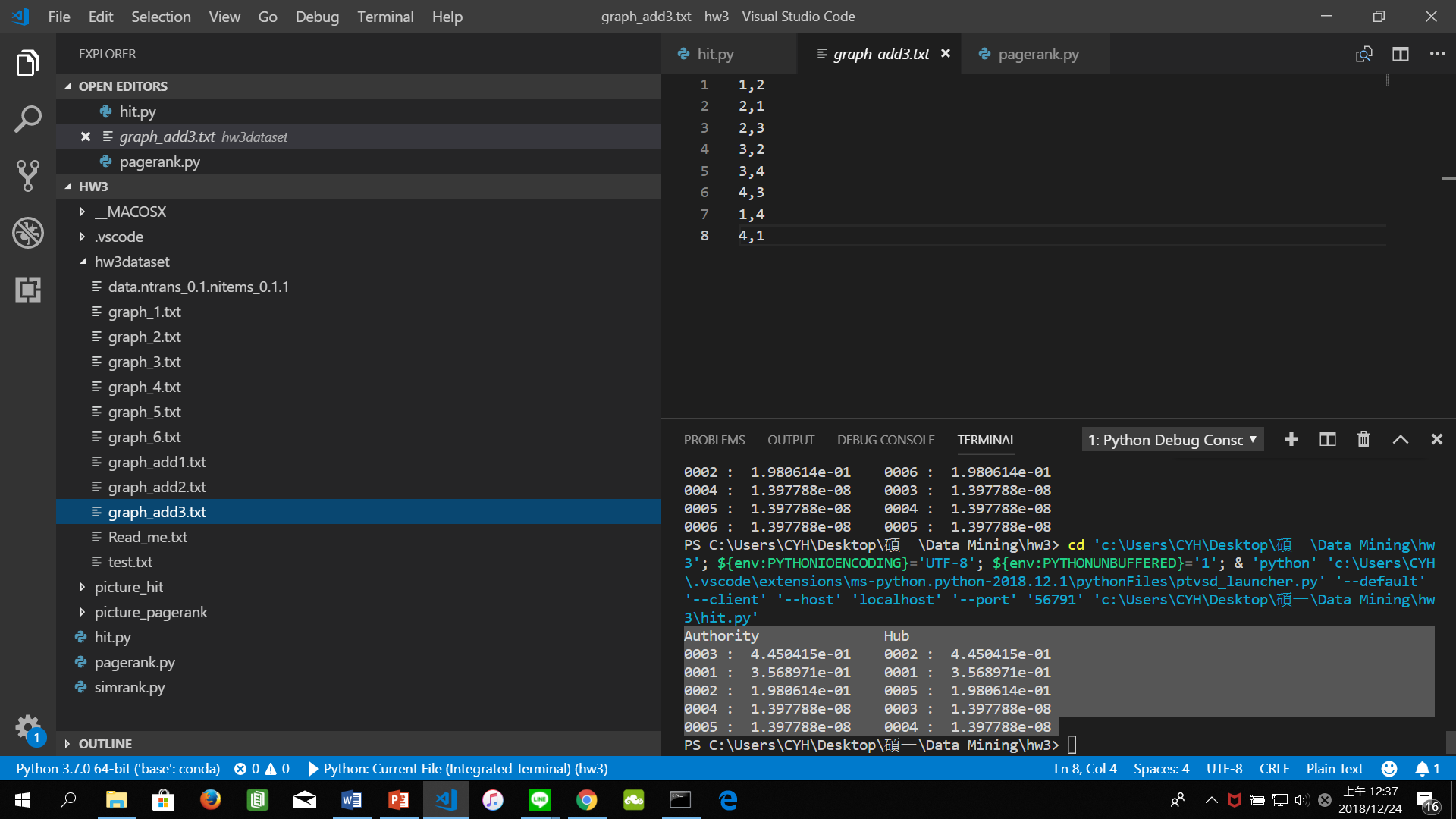
hub 大小順序: 2 3 1 4

authority : 2 3 相同且大是因為都有兩個node指向他們

hub : 2 3 相同且大是因為他們都有指向兩個node

而在這中，可以發現 authority 跟 hub 相同是因為是雙向的

**修改graph3: graph\_add3.txt**

 -> 

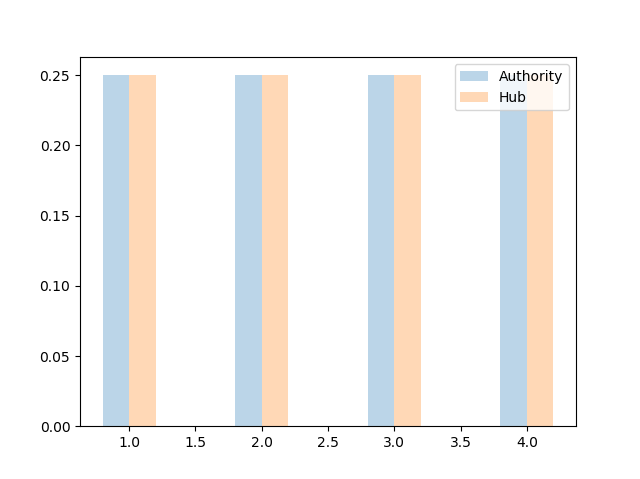
Authority Hub

0001 : 2.500000e-01 0001 : 2.500000e-01

0002 : 2.500000e-01 0002 : 2.500000e-01

0003 : 2.500000e-01 0003 : 2.500000e-01

0004 : 2.500000e-01 0004 : 2.500000e-01

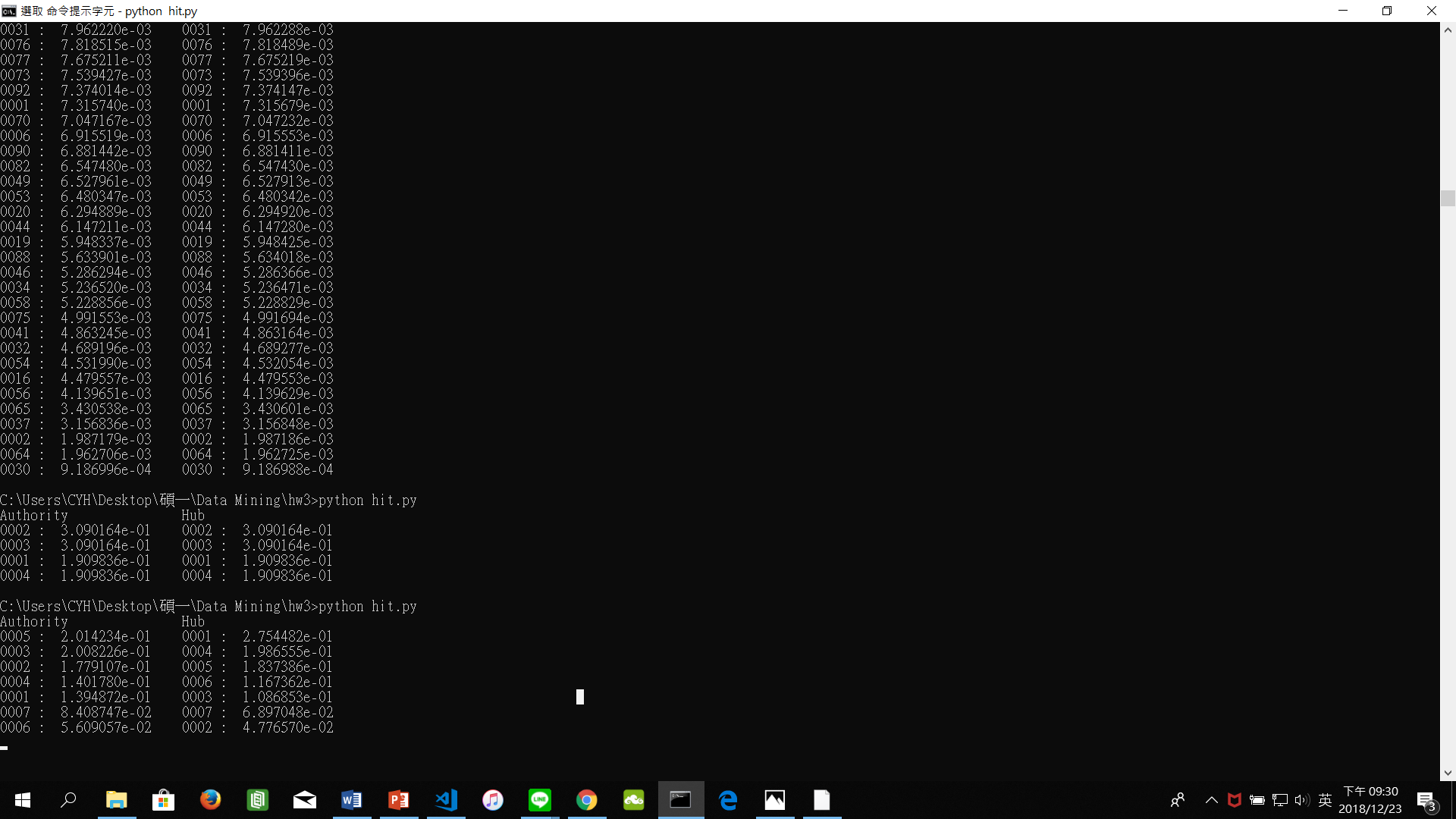


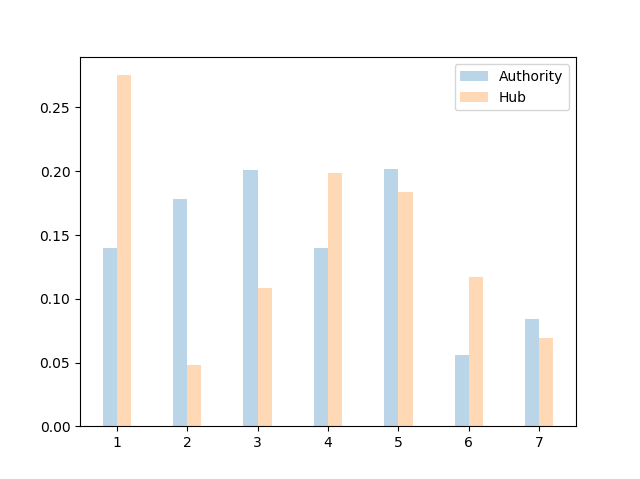
原本 1 的Authority: 1.909836e-01 Hub: 1.909836e-01

後來 1 的Authority: 2.500000e-01 Hub: 2.500000e-01

都有增加

1. graph 4





authority 大小順序 : 5 3 2 4 1 7 6

hub 大小順序 : 1 4 5 6 3 7 2

authority : 5 最大是因為都有最多的node指向他

hub : 1 最大是因為他指向最多個node

1. graph 5

**輸出的前面幾行:**

Authority Hub

0061 : 9.584816e-02 0274 : 2.823663e-02

0122 : 9.415026e-02 0176 : 2.792339e-02

0212 : 5.756808e-02 0412 : 2.732402e-02

0104 : 5.592685e-02 0293 : 2.612433e-02

**輸出的中間還有非常非常多行，但因為全部貼到word上會佔太多頁，所以就只貼前面和後面的輸出**

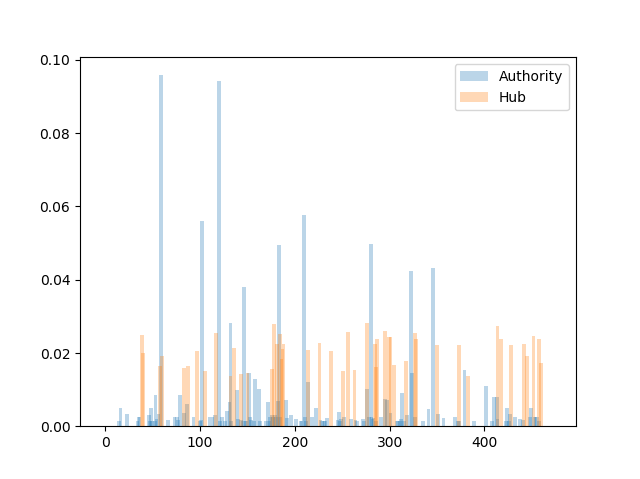
**輸出的後面幾行:**

0002 : 0.000000e+00 0465 : 0.000000e+00

0003 : 0.000000e+00 0466 : 0.000000e+00

0004 : 0.000000e+00 0467 : 0.000000e+00

0005 : 0.000000e+00 0469 : 0.000000e+00



1. graph 6

**輸出的前面幾行:**

Authority Hub

Authority Hub

0761 : 3.039952e-02 0171 : 1.614977e-02

1151 : 3.039952e-02 0857 : 1.551735e-02

0062 : 3.017349e-02 0185 : 1.541654e-02

0078 : 3.002696e-02 0091 : 1.528986e-02

0394 : 2.931671e-02 0079 : 1.524725e-02

0863 : 2.862263e-02 1199 : 1.506761e-02

1123 : 2.819881e-02 0499 : 1.504076e-02

**輸出的中間還有非常非常多行，但因為全部貼到word上會佔太多頁，所以就只貼前面和後面的輸出**

**輸出的後面幾行:**

0854 : 7.220368e-97 1221 : 0.000000e+00

0901 : 7.220368e-97 1222 : 0.000000e+00

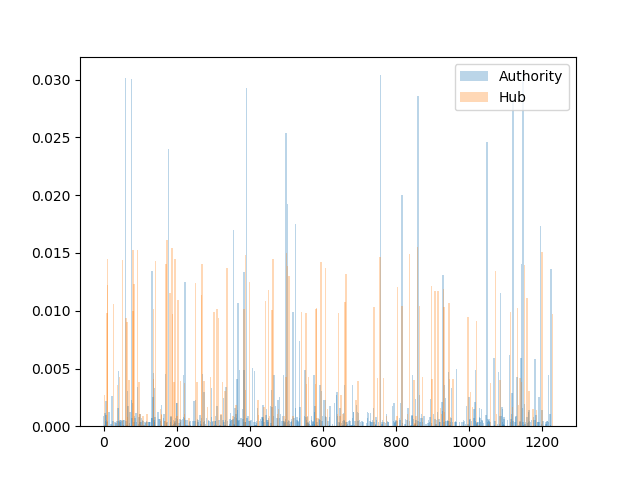
0908 : 3.194175e-128 1223 : 0.000000e+00

1120 : 3.194175e-128 1224 : 0.000000e+00

0051 : 1.134799e-142 1225 : 0.000000e+00

0358 : 1.134799e-142 1226 : 0.000000e+00

0001 : 0.000000e+00 1228 : 0.000000e+00



1. (directed)

**輸出的前面幾行:**

Authority Hub

0038 : 4.791469e-02 0091 : 2.197282e-02

0063 : 3.517298e-02 0083 : 1.980840e-02

0036 : 3.146717e-02 0055 : 1.855304e-02

0087 : 2.965093e-02 0079 : 1.623587e-02

0048 : 2.854402e-02 0060 : 1.613494e-02

0069 : 2.702897e-02 0048 : 1.578159e-02

0085 : 2.303761e-02 0024 : 1.569716e-02

**輸出的中間還有非常非常多行，但因為全部貼到word上會佔太多頁，所以就只貼前面和後面的輸出**

**輸出的後面幾行:**

0002 : 0.000000e+00 0033 : 4.037052e-03

0030 : 0.000000e+00 0034 : 3.838807e-03

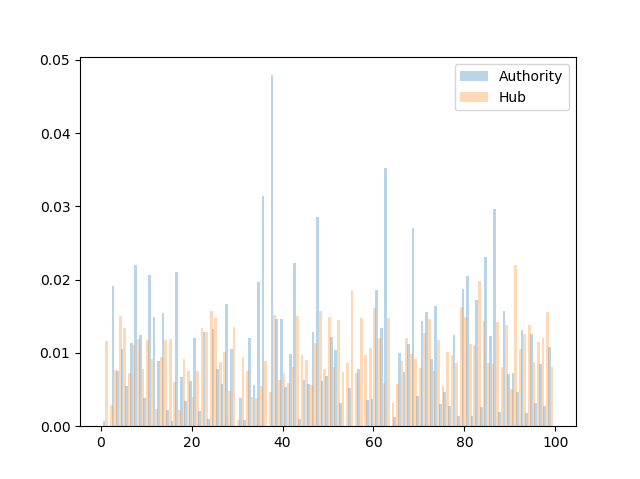
0037 : 0.000000e+00 0064 : 3.192035e-03

0054 : 0.000000e+00 0002 : 2.940754e-03

0056 : 0.000000e+00 0012 : 2.421408e-03

0058 : 0.000000e+00 0017 : 2.229063e-03

0064 : 0.000000e+00 0030 : 8.648483e-04



1. (bidirected)

**輸出的前面幾行:**

Authority Hub

0038 : 2.534756e-02 0038 : 2.534756e-02

0063 : 2.139917e-02 0063 : 2.139917e-02

0048 : 1.922539e-02 0048 : 1.922539e-02

0036 : 1.831164e-02 0036 : 1.831164e-02

0083 : 1.739916e-02 0083 : 1.739916e-02

0087 : 1.736305e-02 0087 : 1.736305e-02

0069 : 1.697852e-02 0069 : 1.697852e-02

**輸出的中間還有非常非常多行，但因為全部貼到word上會佔太多頁，所以就只貼前面和後面的輸出**

**輸出的後面幾行:**

0016 : 4.479557e-03 0016 : 4.479557e-03

0056 : 4.139651e-03 0056 : 4.139651e-03

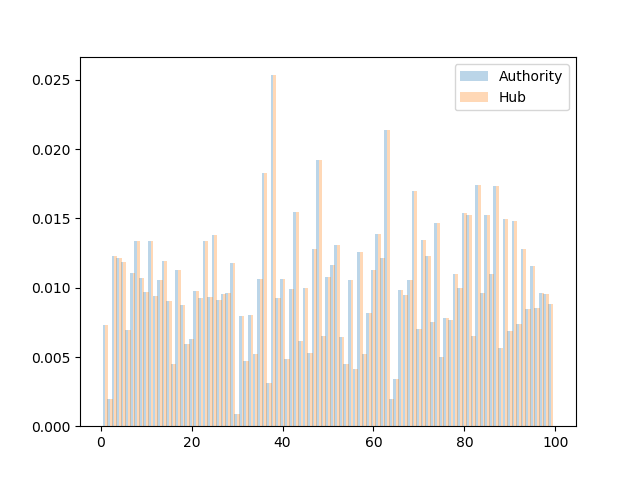
0065 : 3.430538e-03 0065 : 3.430538e-03

0037 : 3.156836e-03 0037 : 3.156836e-03

0002 : 1.987179e-03 0002 : 1.987179e-03

0064 : 1.962706e-03 0064 : 1.962706e-03

0030 : 9.186996e-04 0030 : 9.186996e-04



結果討論:

HITS算法整體而言是個效果很好的算法，目前不僅應用在搜索引擎領域，而且被“自然語言處理”以及“社交分析”等很多其它計算機領域借鑒使用，並取得了很好的應用效果。然而歸納起來，HITS算法還有以下幾個方面存在不足：

1. 計算效率較低

因為HITS算法是與查詢相關的算法，所以必須在接收到用戶查詢後實時進行計算，而HITS算法本身需要進行很多輪迭代計算才能獲得最終結果，這導致其計算效率較低，這是實際應用時必須慎重考慮的問題。

1. 主題漂移問題

如果在擴展網頁集合裡包含部分與查詢主題無關的頁面，而且這些頁面之間有較多的相互鏈接指向，那麼使用HITS算法很可能會給予這些無關網頁很高的排名，導致搜索結果發生主題漂移，這種現像被稱為“緊密鏈接社區現象”（Tightly-Knit CommunityEffect）。

1. 易被作弊者操縱結果

HITS從機制上很容易被作弊者操縱，比如作弊者可以建立一個網頁，頁面內容增加很多指向高質量網頁或者著名網站的網址，這就是一個很好的Hub頁面，之後作弊者再將這個網頁鏈接指向作弊網頁，於是可以提升作弊網頁的Authority得分。

1. 結構不穩定

所謂結構不穩定，就是說在原有的“擴充網頁集合”內，如果添加刪除個別網頁或者改變少數鏈接關係，則HITS算法的排名結果就會有非常大的改變。

Page Rank

PageRank是Google用於用來標識網頁的等級/重要性的一種方法，是Google用來衡量一個網站的好壞的唯一標準。在揉合了諸如Title標識和Keywords標識等所有其它因素之後，Google通過PageRank來調整結果，使那些更具“等級/重要性”的網頁在搜索結果中另網站排名獲得提升，從而提高搜索結果的相關性和質量。其級別從0到10級，10級為滿分。 PR值越高說明該網頁越受歡迎（越重要）。例如：一個PR值為1的網站表明這個網站不太具有流行度，而PR值為7到10則表明這個網站非常受歡迎（或者說極其重要）。一般PR值達到4，就算是一個不錯的網站了。 Google把自己的網站的PR值定到10，這說明Google這個網站是非常受歡迎的，也可以說這個網站非常重要。

page rank 簡單來說:

        就是一個頁面的得票數由所有鏈向它的頁面的重要性來決定，到一個頁面的超鏈接相當於對該頁投一票。一個頁面的PageRank是由所有鏈向它的頁面（鏈入頁面）的重要性經過遞歸算法得到的。一個有較多鏈入的頁面會有較高的等級，相反如果一個頁面沒有任何鏈入頁面，那麼它沒有等級。

對於某個互聯網網頁A來說，該網頁PageRank的計算基於以下兩個基本假設：

數量假設：

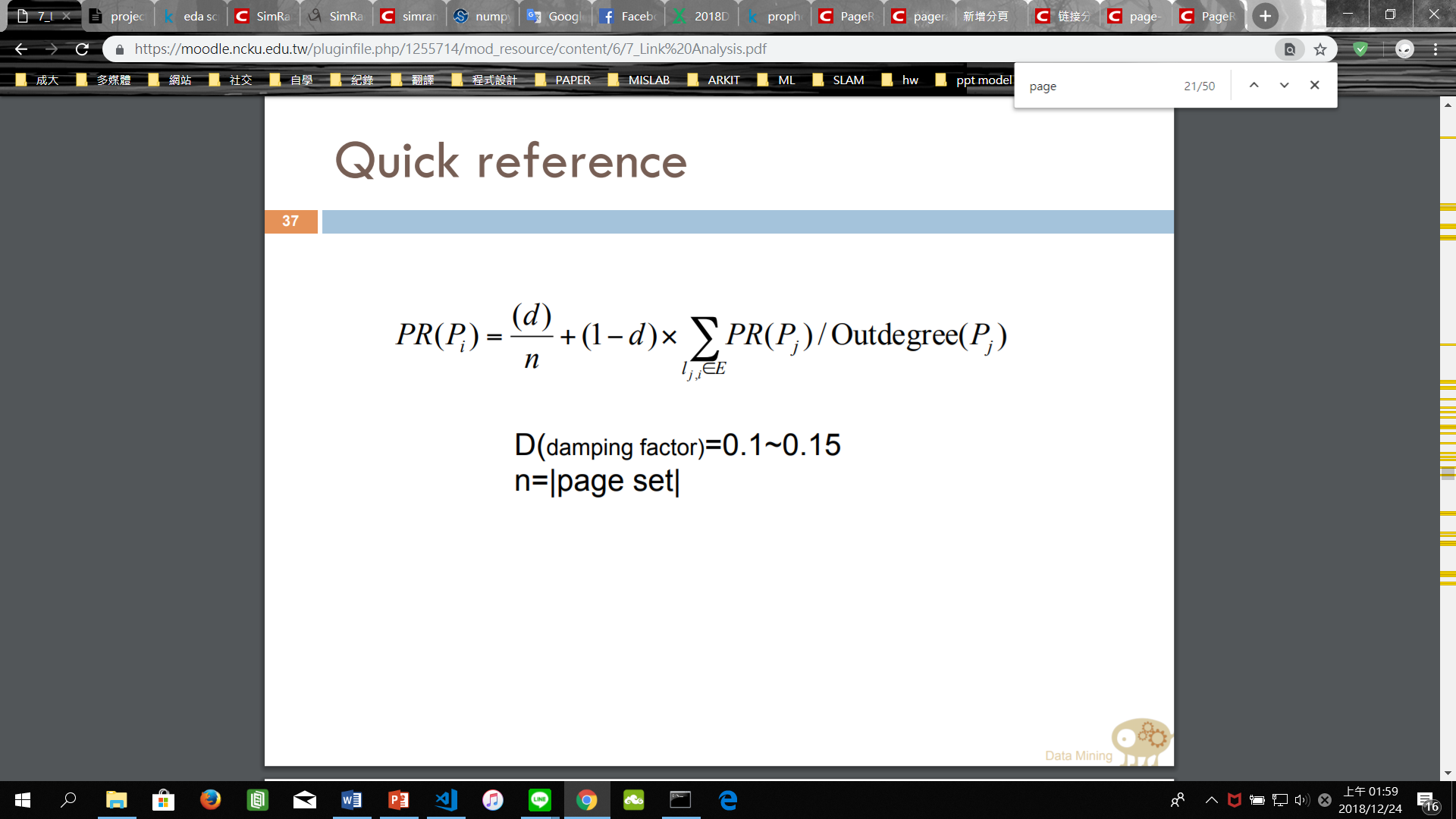
在Web圖模型中，如果一個頁面節點接收到的其他網頁指向的入鏈數量越多，那麼這個頁面越重要。

質量假設：

指向頁面A的入鏈質量不同，質量高的頁面會通過鏈接向其他頁面傳遞更多的權重。所以越是質量高的頁面指向頁面A，則頁面A越重要。

       利用以上兩個假設，PageRank算法剛開始賦予每個網頁相同的重要性得分，通過迭代遞歸計算來更新每個頁面節點的PageRank得分，直到得分穩定為止。 PageRank計算得出的結果是網頁的重要性評價，這和用戶輸入的查詢是沒有任何關係的，即算法是主題無關的。假設有一個搜索引擎，其相似度計算函數不考慮內容相似因素，完全採用PageRank來進行排序，那麼這個搜索引擎的表現是什麼樣子的呢？這個搜索引擎對於任意不同的查詢請求，返回的結果都是相同的，即返回PageRank值最高的頁面。

實作細節:



def pagerank(a, x):

r = np.dot(a,x)

r = r / np.sum(r)

e = 1

while (e > 0.00001):

past\_r = r

r = np.dot(a,r)

r = r / np.sum(r)

e = np.linalg.norm(r-past\_r)

return r

ee = ee/count

a = 0.85 \* pt + 0.15 \* ee

x = np.ones(count)

a 是 使用上圖公式，並帶入 d=0.15

r 是 各個頁面的 PageRank 值

e 是 誤差

公式:

r = np.dot(a,r)

直到 ||r-past\_r|| < e 為止，才停止

實作結果分析:

1. graph 1

Page rank

0006 : 3.314211e-01

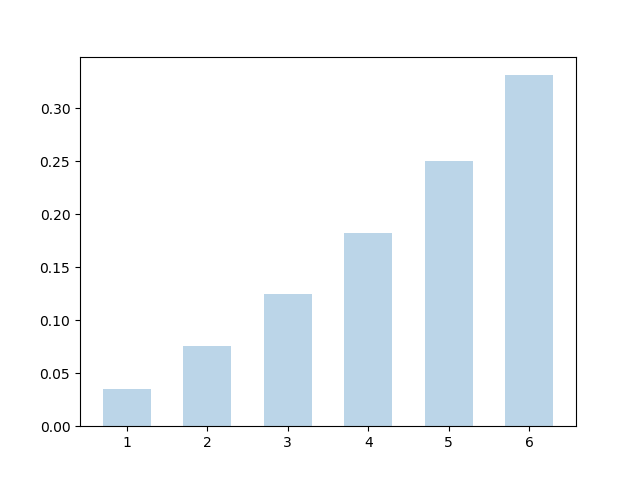
0005 : 2.506510e-01

0004 : 1.824019e-01

0003 : 1.247292e-01

0002 : 7.599177e-02

0001 : 3.480503e-02



page rank 大小是: 6 5 4 3 2 1

這個graph 是單向傳遞

1 -> 2

2 -> 3

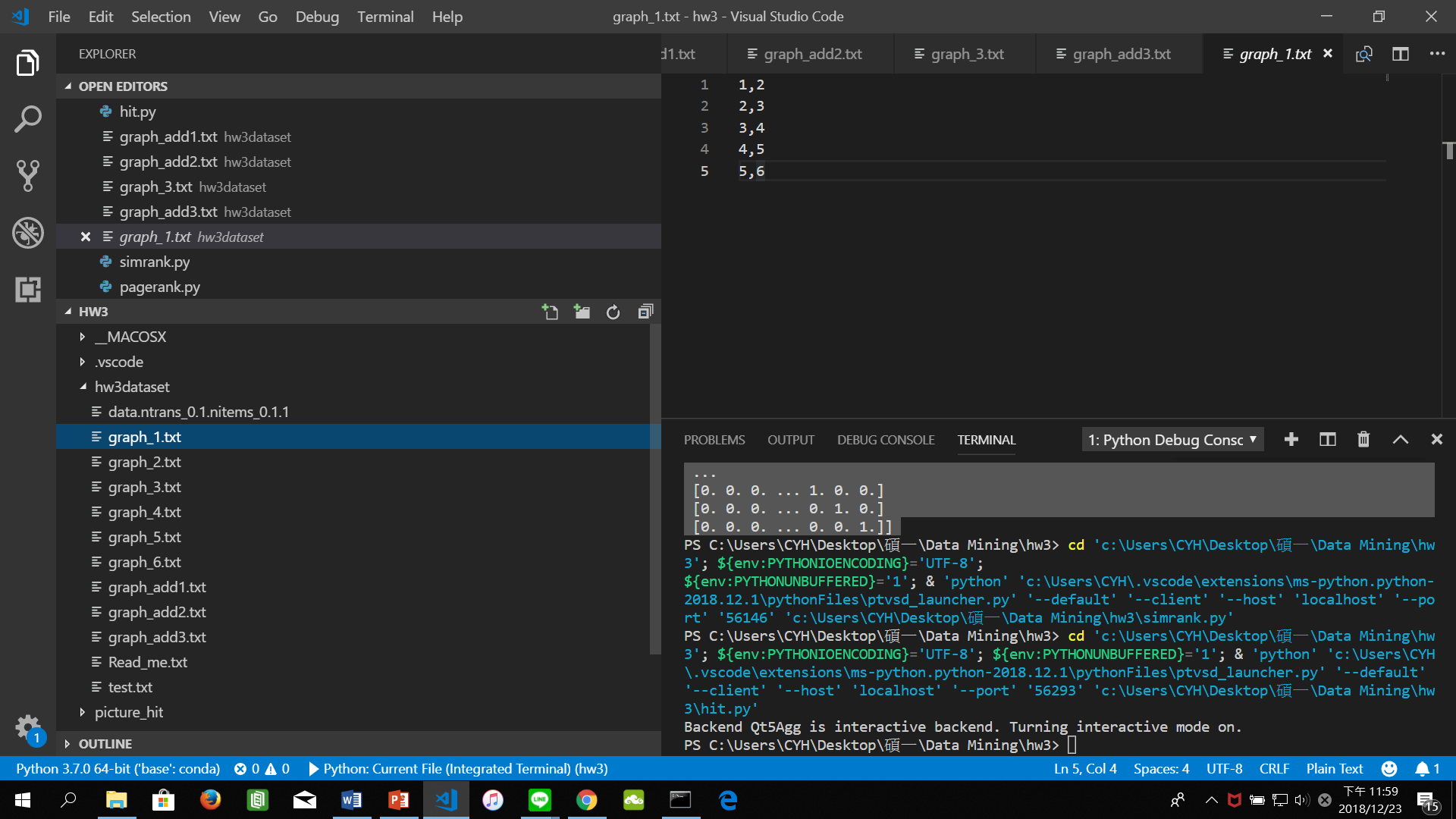
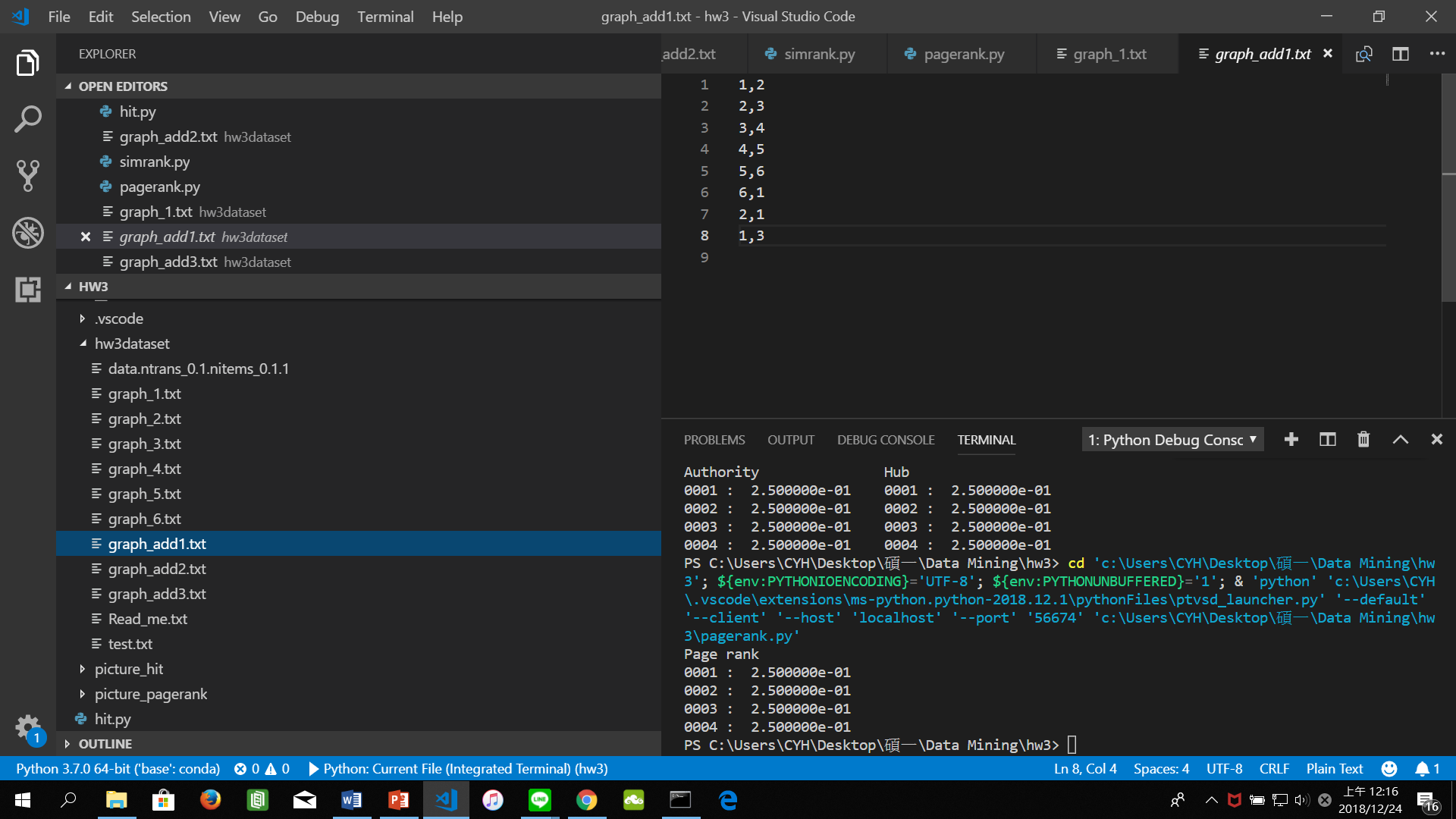
3 -> 4

4 -> 5

5 -> 6

所以依照page rank的計算方式，6的值是會最大的

**修改graph1: graph\_add1.txt**

 -> 

Page rank

0001 : 2.163725e-01

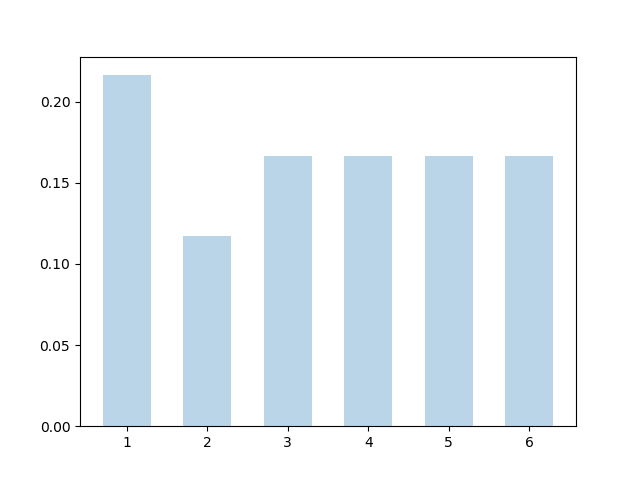
0003 : 1.666667e-01

0006 : 1.666667e-01

0004 : 1.666667e-01

0005 : 1.666667e-01

0002 : 1.169608e-01



page rank 大小是: 1 3 4 5 6 2

1 會變大的原因是因為原本最大的6指向了1，使這個 graph 也變成了迴圈，因此 1 不在是最底階層的，所以page rank值增加

1. graph 2

Page rank

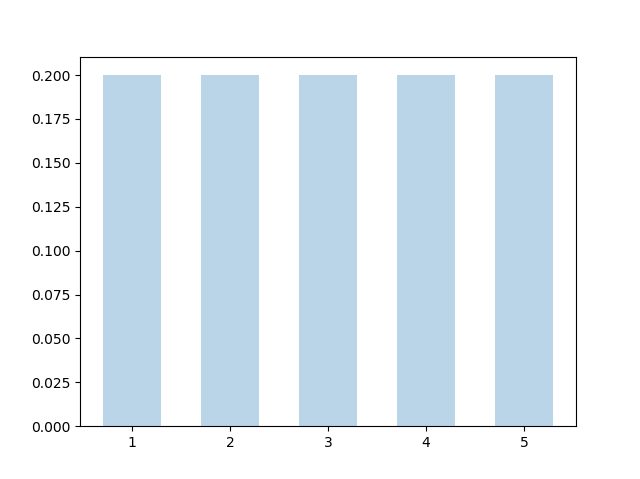
0001 : 2.000000e-01

0002 : 2.000000e-01

0003 : 2.000000e-01

0004 : 2.000000e-01

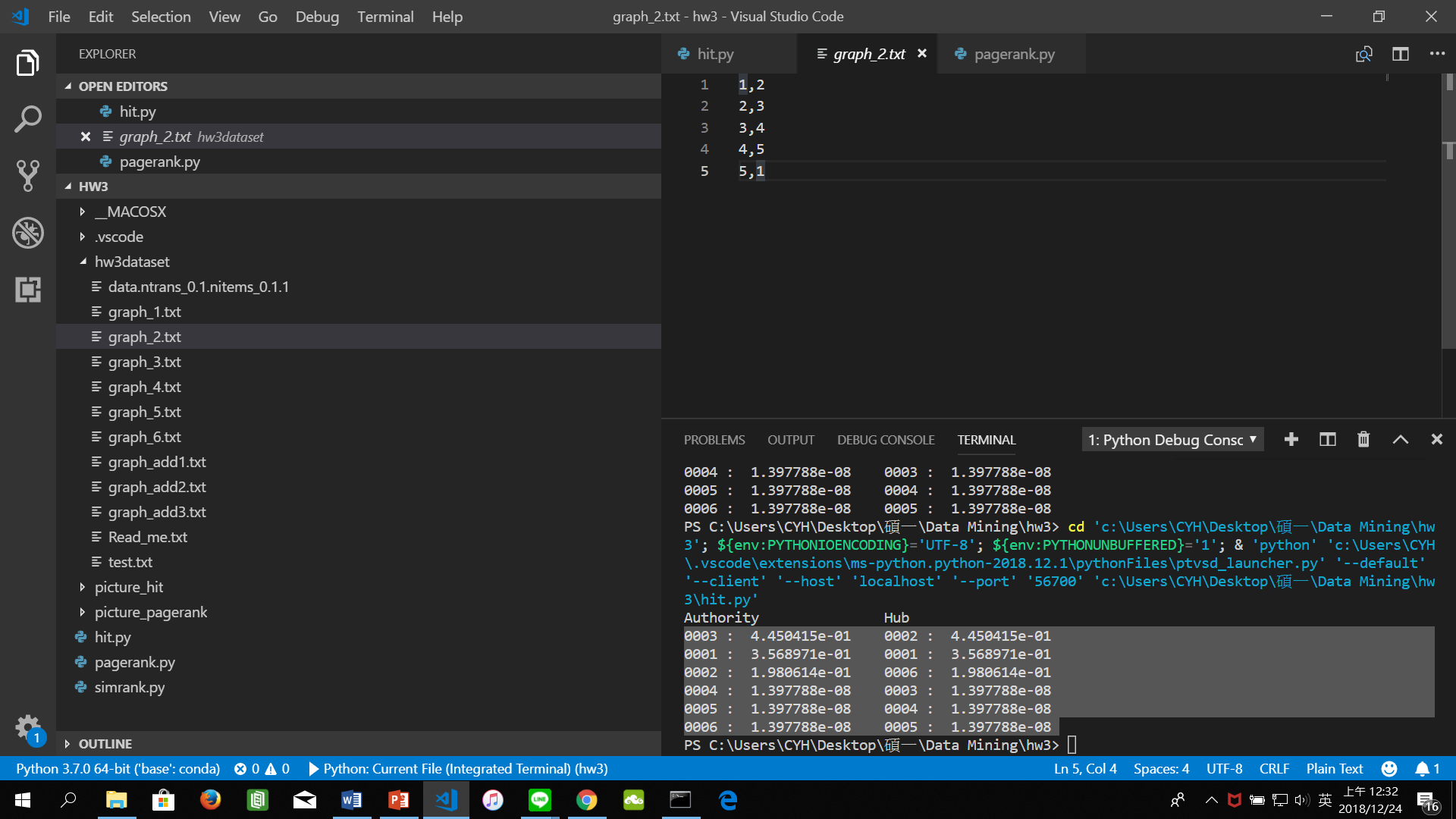
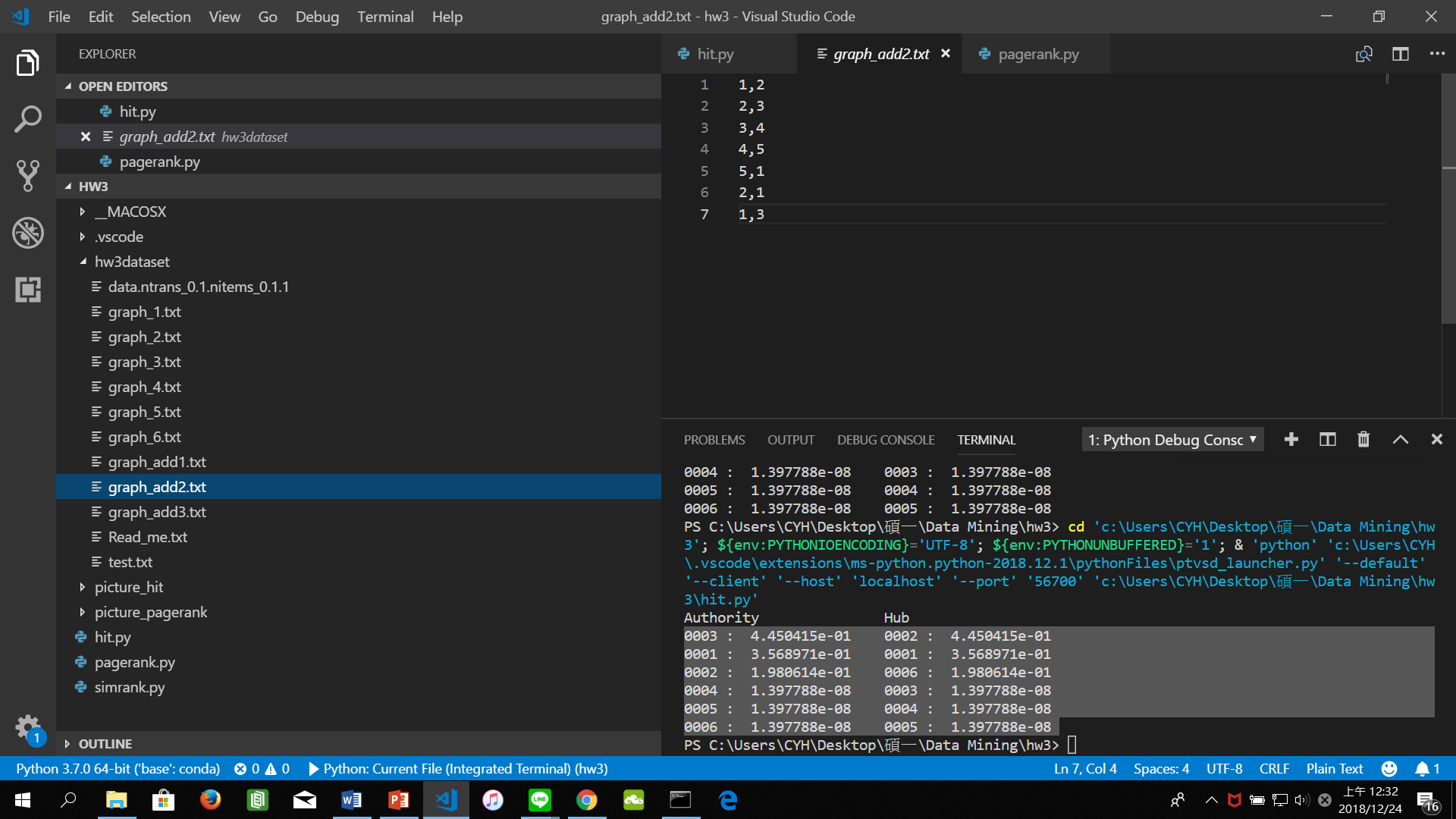
0005 : 2.000000e-01



page rank 大小是: 1 2 3 4 5

都相同的原因是這個 graph 也是一個漂亮沒分岔的迴圈，因此 page rank 相同

**修改graph2: graph\_add2.txt**

 **->** 

Page rank

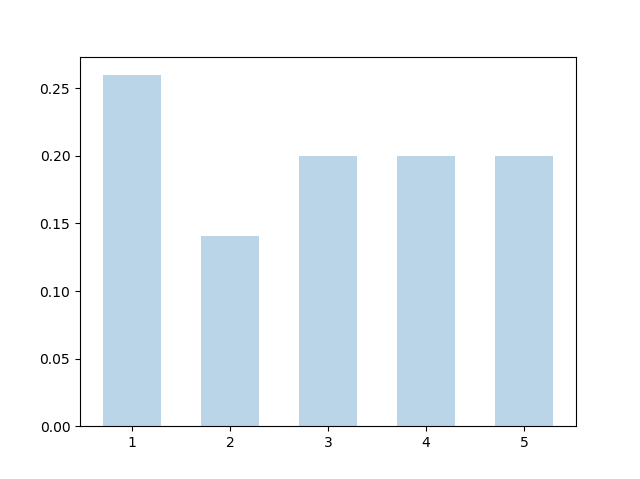
0001 : 2.596471e-01

0003 : 2.000000e-01

0004 : 2.000000e-01

0005 : 2.000000e-01

0002 : 1.403529e-01



page rank 大小是: 1 3 4 5 2

1 會變大的原因是因為多了2指向1，所以page rank值增加

1. graph 3

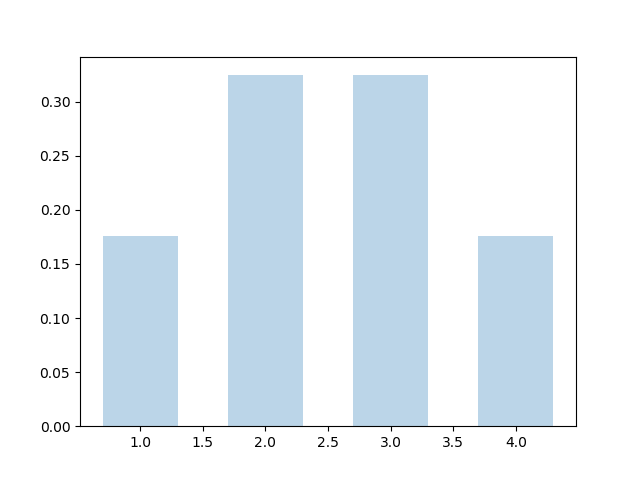
Page rank

0002 : 3.245625e-01

0003 : 3.245625e-01

0001 : 1.754375e-01

0004 : 1.754375e-01



page rank 大小是: 2 3 1 4

從下圖可以很明顯知道 2 3 會相同，1 4會相同

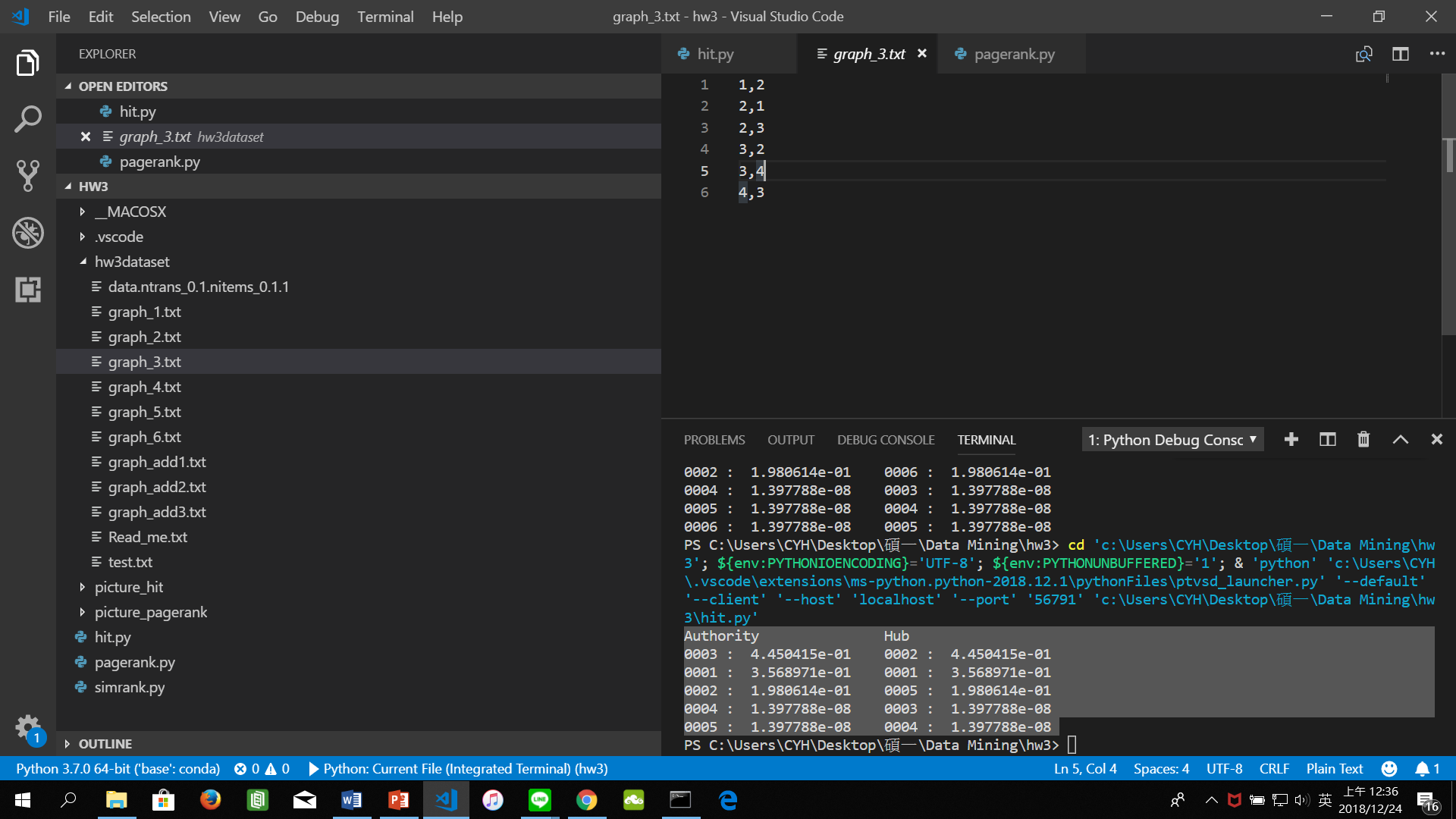
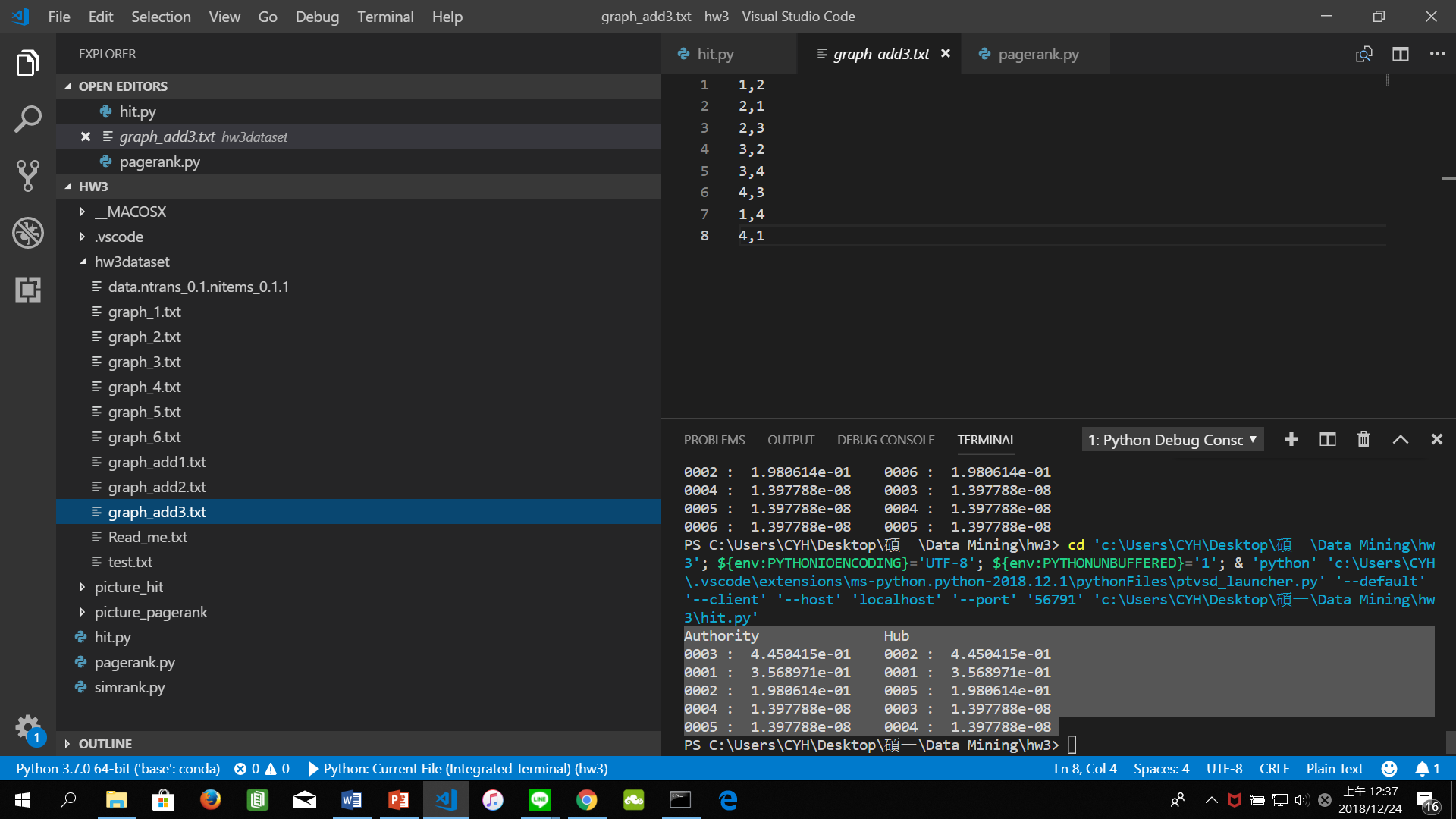
1 -> 2 3 -> 4

1 <- 2 3 -> 4

2 -> 3

2 <- 3

**修改graph3: graph\_add3.txt**

 -> 

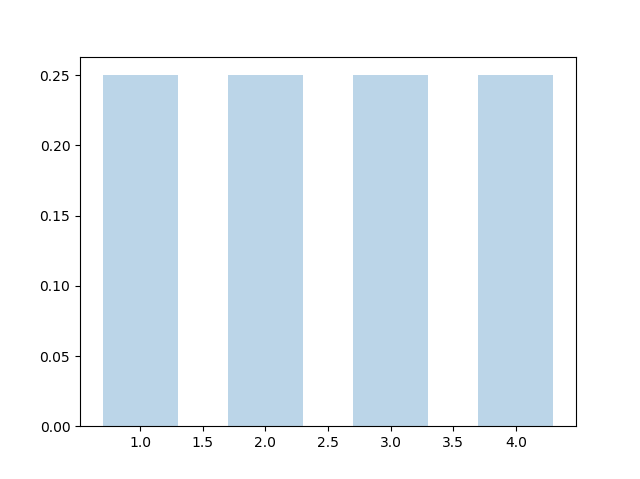
Page rank

0001 : 2.500000e-01

0002 : 2.500000e-01

0003 : 2.500000e-01

0004 : 2.500000e-01



page rank 大小是: 1 2 3 4

都相同的原因是這個 graph 也是一個漂亮沒分岔的迴圈，因此 page rank 相同

1 -> 4

1 <- 4

1 -> 2 3 -> 4

1 <- 2 3 -> 4

2 -> 3

2 <- 3

1. graph 4

Page rank

0001 : 2.802871e-01

0005 : 1.841973e-01

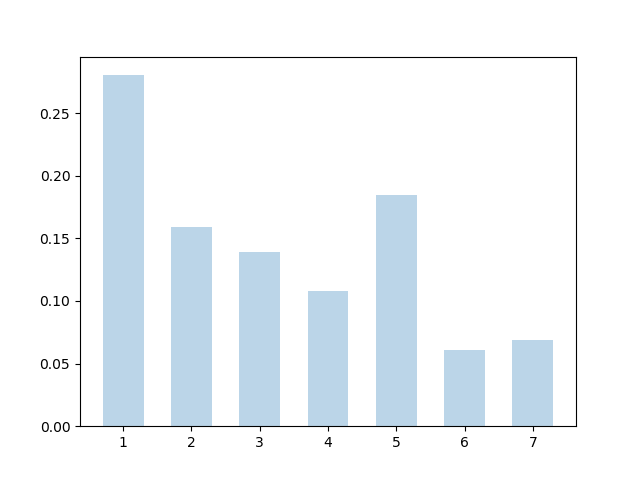
0002 : 1.587644e-01

0003 : 1.388822e-01

0004 : 1.082202e-01

0007 : 6.907804e-02

0006 : 6.057074e-02



page rank 大小是: 1 5 2 3 4 7 6

1. graph 5

**輸出的前面幾行:**

Page rank

0096 : 4.792755e-02

0044 : 3.779602e-02

0024 : 3.435345e-02

0094 : 3.156514e-02

**輸出的中間還有非常非常多行，但因為全部貼到word上會佔太多頁，所以就只貼前面和後面的輸出**

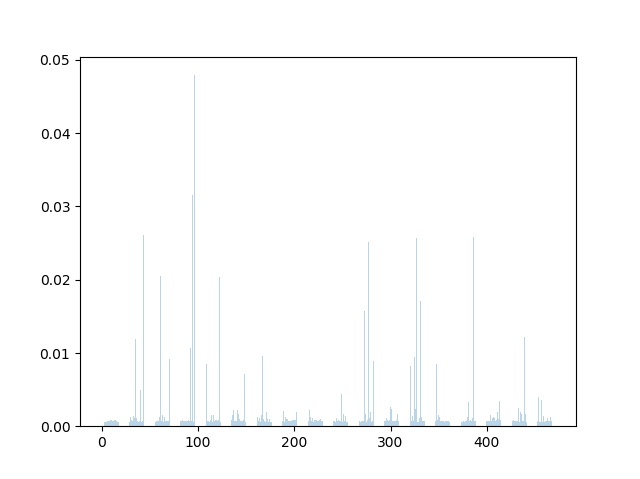
**輸出的後面幾行:**

0002 : 5.942408e-04

0003 : 5.942408e-04

0004 : 5.942408e-04

0005 : 5.942408e-04



page rank 大小 ，從上圖很清楚看到 96 最大

1. graph 6

**輸出的前面幾行:**

Page rank

1052 : 1.553515e-02

0410 : 1.172806e-02

0374 : 1.105869e-02

0387 : 1.105869e-02

**輸出的中間還有非常非常多行，但因為全部貼到word上會佔太多頁，所以就只貼前面和後面的輸出**

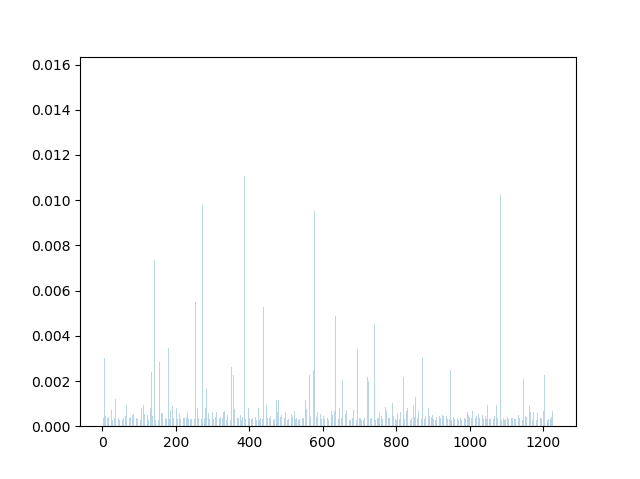
**輸出的後面幾行:**

1175 : 2.838704e-04

1211 : 2.838704e-04

1214 : 2.838704e-04

0001 : 2.696907e-04



1. (directed)

**輸出的前面幾行:**

Page rank

0038 : 4.636807e-02

0063 : 3.729163e-02

0036 : 2.547771e-02

0083 : 2.440551e-02

**輸出的中間還有非常非常多行，但因為全部貼到word上會佔太多頁，所以就只貼前面和後面的輸出**

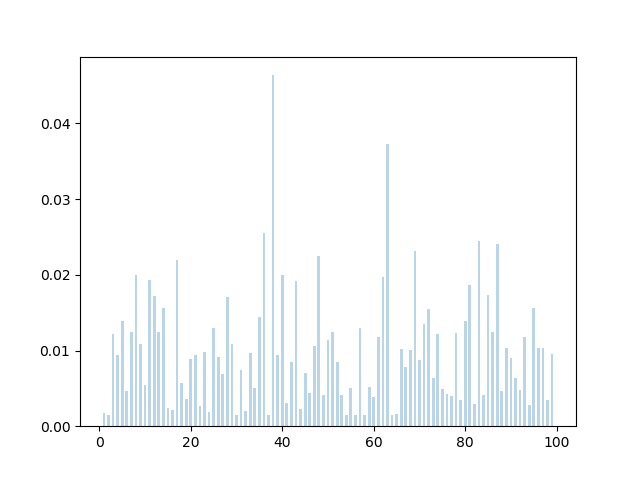
**輸出的後面幾行:**

0054 : 1.515152e-03

0056 : 1.515152e-03

0058 : 1.515152e-03

0064 : 1.515152e-03



1. (bi-directed)

**輸出的前面幾行:**

Page rank

0038 : 2.308657e-02

0063 : 1.926795e-02

0069 : 1.836606e-02

0087 : 1.720205e-02

**輸出的中間還有非常非常多行，但因為全部貼到word上會佔太多頁，所以就只貼前面和後面的輸出**

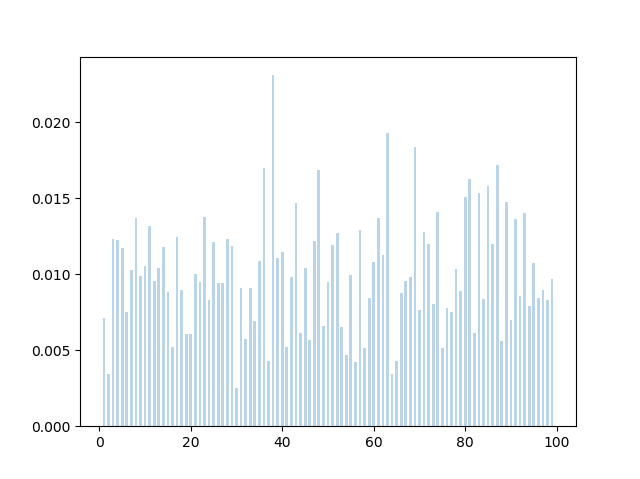
**輸出的後面幾行:**

0056 : 4.216207e-03

0002 : 3.442178e-03

0064 : 3.436864e-03

0030 : 2.507573e-03



結果討論:

PageRank

優點：

        是一個與查詢無關的靜態算法，所有網頁的PageRank值通過離線計算獲得；有效減少在線查詢時的計算量，極大降低了查詢響應時間。

缺點：

1. 人們的查詢具有主題特徵，PageRank忽略了主題相關性，導致結果的相關性和主題性降低
2. 舊的頁面等級會比新頁面高。因為即使是非常好的新頁面也不會有很多上游鏈接，除非它是某個站點的子站點。
3. 沒有區分站內導航鏈接。很多網站的首頁都有很多對站內其他頁面的鏈接，稱為站內導航鏈接。這些鏈接與不同網站之間的鏈接相比，肯定是後者更能體現PageRank值的傳遞關係。
4. 第沒有過濾廣告鏈接和功能鏈接（例如常見的“分享到微博”）。這些鏈接通常沒有什麼實際價值，前者鏈接到廣告頁面，後者常常鏈接到某個社交網站首頁。

HITS與PageRank比較

     HITS算法和PageRank算法可以說是搜索引擎鏈接分析的兩個最基礎且最重要的算法。從以上對兩個算法的介紹可以看出，兩者無論是在基本概念模型還是計算思路以及技術實現細節都有很大的不同。

    1.HITS算法是與用戶輸入的查詢請求密切相關的，而PageRank與查詢請求無關。所以，HITS算法可以單獨作為相似性計算評價標準，而PageRank必須結合內容相似性計算才可以用來對網頁相關性進行評價；

2.HITS算法因為與用戶查詢密切相關，所以必須在接收到用戶查詢後實時進行計算，計算效率較低；而PageRank則可以在爬蟲抓取完成後離線計算，在線直接使用計算結果，計算效率較高；

    3.HITS算法的計算對像數量較少，只需計算擴展集合內網頁之間的鏈接關係；而PageRank是全局性算法，對所有互聯網頁面節點進行處理；

    4.從兩者的計算效率和處理對象集合大小來比較，PageRank更適合部署在服務器端，而HITS算法更適合部署在客戶端；

    5.HITS算法存在主題泛化問題，所以更適合處理具體化的用戶查詢；而PageRank在處理寬泛的用戶查詢時更有優勢；

6.HITS算法在計算時，對於每個頁面需要計算兩個分值，而PageRank只需計算一個分值即可；在搜索引擎領域，更重視HITS算法計算出的Authority權值，但是在很多應用HITS算法的其它領域，Hub分值也有很重要的作用；

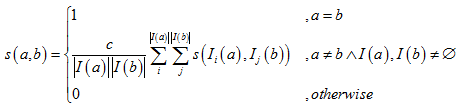
    7.從鏈接反作弊的角度來說，PageRank從機制上優於HITS算法，而HITS算法更易遭受鏈接作弊的影響。

8.HITS算法結構不穩定，當對“擴充網頁集合”內鏈接關係作出很小改變，則對最終排名有很大影響；而PageRank相對HITS而言表現穩定，其根本原因在於PageRank計算時的“遠程跳轉”。

Sim Rank

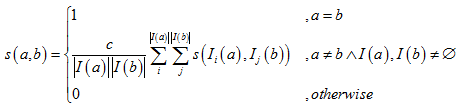
SimRank算法是一種用於衡量結構上下文中個體相似度的方法，其基本思想是：如果兩個對象a和b分別與另外兩個對象c和d關聯，且已知c與d是相似的，則a與b也是相似的；並且任意節點與其自身擁有最大的相似度值為1。 SimRank算法的主要出發點是利用已有個體的相似度來推算其他與之有關聯個體的相似度。

SimRank算法基於一個簡單和直觀的圖論模型，它把對象和對象之間的關係建模為一個有向圖G = (V, E)，其中V是有向圖的節點集合，代表應用領域中的所有對象；E是有向圖的邊的集合，表示對象間的關係。對於圖中的一個節點a，與其所有入邊關聯的鄰節點集合(in-neighbors)記為I(a)，同時，其出邊對應的鄰節點集合(out-neighbors)集合記為O(a )。用s(a, b)表示對象a和對象b之間的相似性，其計算公式為：

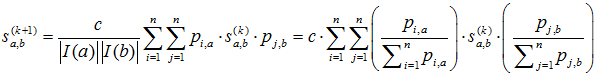


從該計算公式可以看出，個體a, b的相似度取決於所有與a, b相連節點的相似度。式中c是一個常量衰減因子。

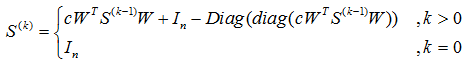
實作細節:



上述公式可以用矩陣的形式表示出來。假設S表示有向圖G的SimRank分數矩陣，其中s(i,j)表示對象i和j之間的相似性分數； P表示G的連接矩陣，其中P(i,j)表示從頂點i到頂點j的邊數，則



用矩陣的符號表示，即為：



其中，矩陣W表示按列歸一化的P矩陣， I是n×n的單位矩陣。對於一個矩陣X，diag(X)表示獲得由所有X的主對角線上的元素構成的向量；對於一個向量x，Diag(x)操作得到相應的對角矩陣，即x = diag(Diag( x))。由於任意對象和自己的相似度值為1，所以加上項

26c78193c9c6478d0e74909889e0d3496a51df9b (241Ã34)

，其作用是把矩陣 的主對角線元素設為1

def simrank(w, s, i\_):

e = 1

while (e > 0.00001):

past\_s = s

s = 0.8 \* np.dot(np.dot(w.T, s),w) + i\_ -np.diag(np.diag(0.8 \* np.dot(np.dot(w.T, s),w)))

e = np.linalg.norm(s-past\_s)

return s

s 是 矩陣，存在 node 和node間的相似性

w 是 轉移概率矩陣，它的每一行和為1，如果從節點i可以轉移到節點j，並且這樣的節點i一共有n個，則Qi,j=1/n

i 是 單位矩陣

e 是 誤差

公式:

s = 0.8 \* np.dot(np.dot(w.T, s),w) + i\_ -np.diag(np.diag(0.8 \* np.dot(np.dot(w.T, s),w)))直到 ||s-past\_s|| < e 為止，才停止

實作結果分析:

1. graph 1

Sim rank

[[1. 0. 0. 0. 0. 0.]

[0. 1. 0. 0. 0. 0.]

[0. 0. 1. 0. 0. 0.]

[0. 0. 0. 1. 0. 0.]

[0. 0. 0. 0. 1. 0.]

[0. 0. 0. 0. 0. 1.]]

1. graph 2

Sim rank

[[1. 0. 0. 0. 0.]

[0. 1. 0. 0. 0.]

[0. 0. 1. 0. 0.]

[0. 0. 0. 1. 0.]

[0. 0. 0. 0. 1.]]

1. graph 3

Sim rank

[[1. 0. 0.66666509 0. ]

[0. 1. 0. 0.66666509]

[0.66666509 0. 1. 0. ]

[0. 0.66666509 0. 1. ]]

1. graph 4

Sim rank

[[1. 0.36026218 0.34895838 0.35373181 0.33765595 0.41507442

0.2923892 ]

[0.36026218 1. 0.40678902 0.36974438 0.41217936 0.28543873

0.45405004]

[0.34895838 0.40678902 1. 0.44956403 0.39005135 0.44809201

0.45103605]

[0.35373181 0.36974438 0.44956403 1. 0.34269175 0.53506185

0.53506185]

[0.33765595 0.41217936 0.39005135 0.34269175 1. 0.27314517

0.41223832]

[0.41507442 0.28543873 0.44809201 0.53506185 0.27314517 1.

0.27012371]

[0.2923892 0.45405004 0.45103605 0.53506185 0.41223832 0.27012371

1. ]]

1. graph 5

這是輸出的樣子，但其中含有非0非1的值

Sim rank

[[1. 0. 0. ... 0. 0. 0.]

[0. 1. 0. ... 0. 0. 0.]

[0. 0. 1. ... 0. 0. 0.]

...

[0. 0. 0. ... 1. 0. 0.]

[0. 0. 0. ... 0. 1. 0.]

[0. 0. 0. ... 0. 0. 1.]]

第一列: [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

第十一列:

[0. 0. 0. 0. 0. 0.

0. 0.8 0. 0. 1. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.4

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0.4 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0.26666667 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.26666667

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0.16 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. ]

結果討論:

SimRank是一種經典的基於結構的相似度算法,具有快速收斂的優點,但其預計算時間和存儲開銷非常巨大,不適用當前大規模和快速增長環境下的Web頁面網絡。

加速SimRank計算的優化技術：

1. 基本節點選擇可以消除具有先驗零分數的一部分節點對的計算。
2. 部分和記憶可以通過高速緩存部分相似性求和以便以後重用來有效地減少不同節點對之間的相似性的重複計算。
3. 對相似性的閾值設置使得能夠進一步減少要計算的節點對的數量。

計算效率分析:

hit

graph1: count: 6 executing hit function time: 9.973049e-04 hit function iteration counts: 1

graph2: count: 5 executing hit function time: 9.984970e-04 hit function iteration counts: 1

graph3: count: 4 executing hit function time: 9.980202e-04 hit function iteration counts: 11

graph4: count: 7 executing hit function time: 9.975433e-04 hit function iteration counts: 19

graph5: count: 469 executing hit function time: 2.912357e-01 hit function iteration counts: 17

graph6: count: 1228 executing hit function time: 7.715178e-02 hit function iteration counts: 95

direct: count: 99 executing hit function time: 2.094507e-02 hit function iteration counts: 11

bi\_direct: count: 99 executing hit function time: 1.293278e-02 hit function iteration counts: 8

add graph1: count: 6 executing hit function time: 9.977818e-04 hit function iteration counts: 28

add graph2: count: 5 executing hit function time: 9.896755e-04 hit function iteration counts: 28

add graph3: count: 4 executing hit function time: 0.000000e+00 hit function iteration counts: 1

pagerank

graph1: count: 6 executing hit function time: 9.980202e-04 hit function iteration counts: 25

graph2: count: 5 executing hit function time: 9.963512e-04 hit function iteration counts: 1

graph3: count: 4 executing hit function time: 9.975433e-04 hit function iteration counts: 12

graph4: count: 7 executing hit function time: 9.977818e-04 hit function iteration counts: 13

graph5: count: 469 executing hit function time: 4.986525e-03 hit function iteration counts: 44

graph6: count: 1228 executing hit function time: 2.593040e-02 hit function iteration counts: 34

direct: count: 99 executing hit function time: 1.097107e-02 hit function iteration counts: 8

bi\_direct: count: 99 executing hit function time: 4.987001e-03 hit function iteration counts: 8

add graph1: count: 6 executing hit function time: 0.000000e+00 hit function iteration counts: 11

add graph2: count: 5 executing hit function time: 0.000000e+00 hit function iteration counts: 11

add graph3: count: 4 executing hit function time: 0.000000e+00 hit function iteration counts: 1

simrank

graph1: count: 6 executing simrank function time: 2.991676e-03 simrank function iteration counts: 2

graph2: count: 5 executing simrank function time: 2.991676e-03 simrank function iteration counts: 2

graph3: count: 4 executing simrank function time: 3.988504e-03 simrank function iteration counts: 12

graph4: count: 7 executing simrank function time: 4.986525e-03 simrank function iteration counts: 26

graph5: count: 469 executing simrank function time: 9.863617e-01 simrank function iteration counts: 34

從上面的資訊看來，執行時間與以下兩點有關:

1. 矩陣大小
2. 幾次迴圈後會收斂

而從上方 hit 、 pagerank 、 simrank 互相作比較，執行時間由快到慢分別為pagerank > hit > simrank

我認為這是因為在迴圈中

pagerank:

r = np.dot(a,r)

hit:

a\_ = np.dot(a.T,past\_h)

h\_ = np.dot(a,past\_a)

sim rank:

s = 0.8 \* np.dot(np.dot(w.T, s),w) + i\_ -np.diag(np.diag(0.8 \* np.dot(np.dot(w.T, s),w)))

pagerank 比 hit快的原因:

少了一行內積處理

hit 比 sim rank快的原因:

sim rank 是矩陣的處理，而Hit 是向量的處理，所以要處理的資料量較小，時間也就比較快速