Project 3_Report

P76074389 蔡雨芝

(Coding with Python 3.6.5, macOS Mojave 10.14.1)

- **執行方式：**python3 HW3.py (must in 'code' folder)
- py 檔中有註解，可以協助了解code內容。
- 輸出結果除了會print出來外，為了方便觀察，另外存在'results'資料夾中，以 graph_(No.)_(Algorithm).txt 為檔名分別存檔。

1. Datasets:

Given Dataset: Graph 1 - 6

- 從.txt 中讀入 links: 整理成 dictionary 的型態，若 Node1 連結到 Node2 及 Node3，儲存成{'1':[2, 3]}的形式。

Dataset from Proj 1:IBM Quest Data Generator dataset [nitems_0.1, ntrans_0.1]

- 採用 Project 1 的 IBM Quest Data(n_items & n_trans set to 100, 方便觀察)
- 改寫為 directed 與 bi-directed 兩種版本，分別為 Graph 7、Graph 8。
- 前處理：將用 Generator 產生之 data 改寫成與 given graphs 一樣格式的.txt 檔。(start_node, end_node)(預先處理好的.txt 檔與 graphs 皆存在 dataset 資料夾中。
- 從.txt 中讀入 links: 整理成 dictionary 的型態，若 Node1 連結到 Node2 及 Node3，儲存成{'1':[2, 3]}的形式。
- 在此 dataset 中，平均 1 個 Node 會連結到的 Node 數 = 10。

```
Graph 1, 2, 3,  4,    5,    6,   7,     8 資料觀察：
#    [5, 5, 4,  7,  468, 1227,  99,   99] -> max_num+1 of each graph
#    [5, 5, 6, 18, 1102, 5220, 967, 1934] -> num of link edges
#    [5, 5, 4,  7,  118,  187,  99,  100] -> num of start nodes
#    [6, 5, 4,  7,  469, 1228, 100,  100] -> num of nodes
```

2. Implement HITS (Graph 1-8)

- Eps = 1e-4
- 利用dictionary的方式處理資料。
- 實作方法：

1. 建立 A_dict(以每個 Node 為 start_node 的 links 構成的 Adjacency Matrix) 與 At_dict(A_dict的Transposed Dictionary)。
2. Initialize Authority & Hub of each node with 1
3. 利用A_dict與Authority相乘後，經過L1 Normalize(sum up to 1)後即為新的Hub值。
4. 利用At_dict與更新後之Hub值相乘後，經過Normalize(sum up to 1)後更新Authority。

5. 重複Step 3 & 4，直到 diff(更新後之Authority, 原本的Authority) + diff(更新後之Hub, 原本的Hub)值 < Eps，收斂後停止loop。

- **Results(Graph 1-4):**

  (Results of all graphs:please refer to results/graph()_hits.txt)

  (備註：{1: [2], 2: [3]}: Node1->Node2, Node2->Node3)

1. Graph 1: {1: [2], 2: [3], 3: [4], 4: [5], 5: [6], 6: []}

>> Authority:		>> Hub:	
2:	0.2	1:	0.2
3:	0.2	2:	0.2
4:	0.2	3:	0.2
5:	0.2	4:	0.2
6:	0.2	5:	0.2
1:	0.0	6:	0.0

2. Graph 2: {1: [2], 2: [3], 3: [4], 4: [5], 5: [1]}

>> Authority:		>> Hub:	
1:	0.2	1:	0.2
2:	0.2	2:	0.2
3:	0.2	3:	0.2
4:	0.2	4:	0.2
5:	0.2	5:	0.2

3. Graph 3: {1: [2], 2: [1, 3], 3: [2, 4], 4: [3]}

>> Authority:		>> Hub:	
2:	0.309018567639257	2:	0.309018567639257
3:	0.309018567639257	3:	0.309018567639257
1:	0.190981432360743	1:	0.190981432360743
4:	0.190981432360743	4:	0.190981432360743

4. Graph 4: {1: [2, 3, 4, 5, 7], 2: [1], 3: [1, 2], 4: [2, 3, 5], 5: [1, 3, 4, 6], 6: [1, 5], 7: [5]}

>> Authority:		>> Hub:	
5:	0.201420258449907	1:	0.275440138466957
3:	0.200821674692092	4:	0.198649161703947
2:	0.177908652297836	5:	0.183744740158434
4:	0.140178538193541	6:	0.116738652442789
1:	0.139492338173083	3:	0.108688654785085
7:	0.0840858618772742	7:	0.0689675054828132
6:	0.0560926763162671	2:	0.0477711469599753

5. Graph 8:(僅列出前7名)

>> Authority:		>> Hub:	
38:	0.0305639580811499	38:	0.0305639580811499
63:	0.0241950552185817	63:	0.0241950552185817
87:	0.0212761754876976	87:	0.0212761754876976
48:	0.0204047097255067	48:	0.0204047097255067
36:	0.0192011258879245	36:	0.0192011258879245
83:	0.0191079732294681	83:	0.0191079732294681
43:	0.0179757287993883	43:	0.0179757287993883

```
Time cost of HITS, G1:  0.0007010 s
Time cost of HITS, G2:  0.0002730 s
Time cost of HITS, G3:  0.0004160 s
Time cost of HITS, G4:  0.0004218 s
Time cost of HITS, G5:  0.0223789 s
Time cost of HITS, G6:  0.1521709 s
Time cost of HITS, G7:  0.0053282 s
Time cost of HITS, G8:  0.0065269 s
(Mean) Time cost of HITS:  0.0235271 s
```

- **討論：**

1. 實際上的HITS是開始於搜尋引擎以Query回饋的網頁，這邊僅實作核心部分(計算Authority & Hub值)。
2. Graph 1中，因為Node 1沒有被任何其他Node指到，所以Authority=0，Node 6因為沒有指向任何Node，因此Hub=0。
3. Graph 2為circle，因此每個Node的Authority與Hub都相同。
4. Graph 3為1⇔2⇔3⇔4，因此邊緣Node(1, 4)之Authority與Hub相同(較低)、中間Node(2, 3)之Authority與Hub相同(較高)，但不是邊緣Node的2倍（因為每次都有Normalize）。
5. Graph 4中，可以看到Node 1是很好的Hub(因為指向的Node較多、且這些Node都是較為重要的Node)，反之雖然5指向的Node也很多，Hub值卻比Node 4還少，這是因為Node 5指向的Node普遍Authority都偏低，因此Hub值也相對受影響而變低。
6. Graph 8 因為是bi-directed，因此各Node的Authority跟Hub值皆會相同。
7. Time Cost 中可以看到，Graph 6之運算時間最長(因為Node數及link數是所有Graph中最多)，Graph 5僅次於 Graph 6（雖然Link數比Graph 8少一些，但Node數是Graph 8的4.7倍，因此可知道Node數對於Time Cost影響是很大的。
8. HITS僅重視指向的網頁數，並沒有另外對於每個網頁的quality做評分，因此可能造成網頁充數「作弊」而達成Authority、Hub很高，而實質上並非反應這個網頁的重要性的狀況。

---

3. Implement PageRank (Graph 1-8)

- Eps = 1e-4 , d = 0.15
- 利用 matrix 的方式處理資料，最後再轉回 dictionary 輸出。
- 實作方法：

1. 建立A_mat(以每個Node為start_node的links構成的Adjacency Matrix)與At_mat(A_mat的Transposed Matrix)、N_mat(N*N的Matrix，N=Node數，所有值初始化為1/N)。
2. R(存放PageRank值之Matrix)初始化為1/N，也就是PageRank值預設=1/N(N=Node數)。
3. 利用公式P = d*N_mat+(1-d)*At_mat，P與R做dot運算以更新R(PageRank值)。
4. 以L1 Normalize R(to sum up to 1)。
5. 不斷重複 Step 3 & 4，直到R與R_prev(上一次的R)差距小於Eps，此時的R即為最終之PageRank值。

- **Results(Graph 1-4):**

  (Results of all graphs:please refer to results/graph_()_page_rank.txt)

1. Graph 1: {1: [2], 2: [3], 3: [4], 4: [5], 5: [6], 6: []}

PageRank of Graph 1: (Sorted by the PageRank value)

Page ID6: 0.331490893127947	Page ID3: 0.124703120251403
Page ID5: 0.250648206535158	Page ID2: 0.0759851790523549
Page ID4: 0.182366091522817	Page ID1: 0.0348065095103204

2. Graph 2: {1: [2], 2: [3], 3: [4], 4: [5], 5: [1]}

PageRank of Graph 2: (Sorted by the PageRank value)

Page ID5: 0.200000000000000	Page ID4: 0.2
Page ID3: 0.200000000000000	Page ID1: 0.2
Page ID2: 0.200000000000000	

3. Graph 3: {1: [2], 2: [1, 3], 3: [2, 4], 4: [3]}

PageRank of Graph 3: (Sorted by the PageRank value)

Page ID3: 0.324547068321367	Page ID4: 0.175452931678633
Page ID2: 0.324547068321367	Page ID1: 0.175452931678633

4. Graph 4: {1: [2, 3, 4, 5, 7], 2: [1], 3: [1, 2], 4: [2, 3, 5], 5: [1, 3, 4, 6], 6: [1, 5], 7: [5]}

PageRank of Graph 4: (Sorted by the PageRank value)

Page ID1: 0.280308893584124	Page ID4: 0.108222541066354
Page ID5: 0.184183727857094	Page ID7: 0.0690712908850486
Page ID2: 0.158753226075398	Page ID6: 0.0605798216098772
Page ID3: 0.138880498922105	

Time cost of PageRank, G1:0.0024867 s	Time cost of PageRank, G6:0.4081223 s
Time cost of PageRank, G2:0.0001926 s	Time cost of PageRank, G7:0.0062640 s
Time cost of PageRank, G3:0.0003822 s	Time cost of PageRank, G8:0.0101922 s
Time cost of PageRank, G4:0.0004528 s	(Mean)Time cost of PageRank:0.0626939 s
Time cost of PageRank, G5:0.0734582 s	

- **討論：**

1. PageRank與HITS的差別為不以Query為主，PageRank以連向此網頁之頁面的「指出的網頁數量」決定分數（重要性），因此HITS適合用戶端查詢使用，PageRank則較適合伺服器端。
2. PageRank 對於所有連結的網頁做計算(不限制於Query)，較HITS更為全面性。
3. 在實際應用上，PageRank不利於新網頁，因為即使新網頁quality很好，仍可能沒有太多網頁連結到此網頁。

4. d的大小影響每次更新時的PageRank值中， sum(Parents指出網頁數量)的重要程度。
5. Graph 1中，雖然Node 6沒有連出至任何頁面，然而它的parent node(Node 5)重要性很高(Node1->2->3->4->5)，因此Node 6相對的就更重要了(Node1->2->3->4->5->6)，因此PageRank最高。
6. Graph 2為circle，因此每個Node的PageRank都相同。
7. Graph 3為1⇔2⇔3⇔4，因此邊緣Node(1, 4)PageRank相同(較低)、中間Node(2, 3)之PageRank相同(較高)，但不是邊緣Node的2倍（因為有Normalize）。
8. Graph 4中，雖然Node6 & 7皆只有一個parent node，但因為Node 5的重要性比Node1還低，因此Node 6的重要性也就比Node 7還低。
9. Time Cost 中可以看到，Graph 6之運算時間最長(因為Node數及link數是所有Graph中最多)，Graph 5僅次於 Graph 6（雖然Link數比Graph 8少一些，但Node數是Graph 8的4.7倍，因此可知道Node數對於Time Cost影響是很大的。

---

4. Implement SimRank (Graph 1-5)

- Eps = 1e-4 , max_iter = 10, C = 0.8
- 利用matrix的方式處理資料，對照link的Adjacency Dictionary，以matrix輸出。
- 實作方法：

1. 建立A_dict(以每個Node為start_node的links構成的Adjacency Dictionary)與At_dict(A_dict的Transposed Dictionary)。
2. 建立ref，存放Node編號的map。
3. 初始化sim(存放SimRank值的matrix)為Identity Matrix (對角線的值是1，其餘為0)。
4. 從A_dict中以iteration跑過每個pair，每次選定2個Node(a, b)作為一個pair，若a=b就跳過。
5. 從At_dict尋找a、b的parent nodes(Ia, Ib)，並計算parent nodes的SimRank值的總和(sum_ab)。
6. 設定sim[ref.index(a)][ref.index(b)]的值為C / (len(Ia) * len(Ib)) * sum_ab。
7. 重複Step 4-6，直到SimRank值收斂(2次SimRank值之誤差<Eps或iteration次數達到max_iter)，輸出sim為最終SimRank值。

- **Results(Graph 1-4):**

(Results of all graphs::please refer to results/graph_()_sim_rank.txt)

1. Graph 1: {1: [2], 2: [3], 3: [4], 4: [5], 5: [6], 6: []}

```
SimRank Matrix of Graph 1:(6, 6)
[[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.]]
```

2. Graph 2: {1: [2], 2: [3], 3: [4], 4: [5], 5: [1]}

```
    SimRank Matrix of Graph 2:(5, 5)
    [[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.]]
```

3. Graph 3: {1: [2], 2: [1, 3], 3: [2, 4], 4: [3]}

```
    SimRank Matrix of Graph 3:(4, 4)
    [[1.         0.         0.66659676 0.        ]
     [0.         1.         0.         0.66659676]
     [0.66659676 0.          1.         0.        ]
     [0.         0.66659676 0.         1.        ]]
```

4. Graph 4: {1: [2, 3, 4, 5, 7], 2: [1], 3: [1, 2], 4: [2, 3, 5], 5: [1, 3, 4, 6], 6: [1, 5], 7: [5]}

```
SimRank Matrix of Graph 4:(7, 7)
[[1.         0.35211472 0.34050956 0.34528816 0.32892511  0.40773299  0.28284333]
 [0.35211472 1.         0.39933231 0.36152279 0.40509544  0.27557619  0.44746939]
 [0.34050956 0.39933231 1.         0.44279155 0.38232967  0.44123837  0.44434473]
 [0.34528816 0.36152279 0.44279155 1.         0.33383297  0.52995841  0.52995841]
 [0.32892511 0.40509544 0.38232967 0.33383297 1.          0.26282791  0.40483802]
 [0.40773299 0.27557619 0.44123837 0.52995841 0.26282791 1.          0.25991683]
 [0.28284333 0.44746939 0.44434473 0.52995841 0.40483802 0.25991683  1.        ]]


Time cost of SimRank, G1:0.0070400 s      Time cost of SimRank, G5:50.5205932 s
Time cost of SimRank, G2:0.0005190 s      (Mean)Time cost of SimRank:10.104119 s
Time cost of SimRank, G3:0.0010221 s
Time cost of SimRank, G4:0.0047259 s
```

- **討論：**

1. 公式中的C越小，SimRank值收斂得越快。以C=0.8與C=0.4來說，參照`results/sim_rank_comparison_c_0_().txt`，Graph 3在C=0.4時僅需6個iteration達到收斂，C=0.8時卻是因為達到max_iter而強制結束。Graph 4在C=0.4時需8個iteration達到收斂，C=0.8時卻一樣因達到max_iter而強制結束。因此C也可以稱為每次更動程度的指標。
2. SimRank與前兩個演算法不同的是，SimRank是計算Node Pair之間的相似度，而非給每個Node一個獨立的分數。另外SimRank不需要Normalize（僅計算sum）。

3. 在實際應用方面，SimRank可以用在提供搜尋引擎一個回饋分數，進而優化搜尋結果。然而因為SimRank需要計算(Node數)^{2}次，時間複雜度較高，若需要計算的Node數很大，就會使SimRank計算的花費時間很長。從Time Cost 中可以看到，Graph 6之運算時間最長(因為Node數及link數是所有Graph中最多)，因此可知Node數對於Time Cost影響是很大的。
4. 一開始聽老師上課講解SimRank時有點疑惑：那如果圖形是具有cycle（如Graph 2），要怎麼樣才能讓SimRank達到收斂呢？（因為上課時舉的例子是沒有這個問題的）後來才想到可以透過設定max_iter的方式讓有這種狀況的graph iteration強制結束。
5. Graph 1中(Node1->2->3->4->5->6)，因為每個Node的parent node都沒有交集，因此輸出的SimRank Matrix = Identity Matrix。
6. Graph 2為circle，每個Node的parent node如Graph 1一樣沒有交集，因此輸出的SimRank Matrix = Identity Matrix。
7. Graph 3為1⇔2⇔3⇔4，因此SimRank Matrix會是一個對稱矩陣。
8. Graph 4中，雖然SimRank看似很像對稱矩陣，其實不然。

   以sim(1, 7)和sim(7, 1)的sum_ab來觀察:

   >> iter=1:

   Node1之parent nodes=[2,3,5,6], Node 7 之parent nodes=[1]，其中[2,3,5,6]有連結到Node1，Node 1連結到[2,3,5]。

   sim(1, 7)=4; sim(7, 1)=3(差別只有Node 6->1有link, Node1->6沒有link)

   >> iter=2:

   sim(**1**,7):Node[2, 3, 5, 6]之parent nodes=[1, 3, 4, 5, 6, 7],

   sim(7,**1**):Node [2, 3, 5]之parent nodes=[1, 3, 4, 5, 6, 7]（兩者相同）;

   sim(1,**7**) & sim(**7**,1):Node[1]之parent nodes=[2, 3, 5, 6]。

   sim(1,7)->sim([2,3,5,6], 1)=4-> sim([1,3,4,5,6,7], [2,3,5,6])=13 -> ...

   sim(7,1)->sim(1, [2,3,5,6])=3-> sim([2,3,5,6] , [1,3,4,5,6,7])=11 -> ...

   後續的iteration中，sim(1, 7)及 sim(7, 1)之sum_ab所增加的值都很接近但不盡相同。因此最終輸出的SimRank看起來數值會幾乎一樣，但實際上檢查後會發現並不相等（只是SimRank值非常相近）。

---

5. Increase Hub/Authority/PageRank of Node1 (Graph 1-3)

- **Links_to_be_added = {'1':[4], '4':[1]}**
- 經過實驗，在Graph 1-3中，若加入{'1':[4], '4':[1]}，Node 1的Hub / Authority / PageRank 值都會增加。
- 實作方法：

  以原本的graph links及加入{'1':[4], '4':[1]}後的links做比較(以下稱為Before & After)。

- **Results & Discussion (Graph 1-3):**

1. Graph 1:

```
    {1: [2], 2: [3], 3: [4], 4: [5], 5: [6], 6: []}
(Before) Graph 1
>> Authority:                                  >> Hub:
    2:  0.2                                      1:  0.2
    3:  0.2                                      2:  0.2
    4:  0.2                                      3:  0.2
    5:  0.2                                      4:  0.2
    6:  0.2                                      5:  0.2
    1:  0.0                                      6:  0.0
-------------------------------------------------
    {1: [2, 4], 2: [3], 3: [4], 4: [5, 1], 5: [6], 6: []}
(After) Graph 1
>> Authority:                                  >> Hub:
    4:  0.617981333404616                        1:  0.618007659955672
    2:  0.381933468457034                        3:  0.381949739160393
    1:  0.0000425990691739755                    4:  0.0000426008839319770
    5:  0.0000425990691739755                    2: 0.00000000000000123984890326325
    3: 0.00000000000000123979608685405           5: 0.00000000000000123984890326325
    6: 0.00000000000000123979608685405           6:  0.0
=================================================
    {1: [2], 2: [3], 3: [4], 4: [5], 5: [6], 6: []}
(Before) PageRank of Graph 1: (Sorted by the PageRank value)
    Page ID6:   0.331490893127947                Page ID3:   0.124703120251403
    Page ID5:   0.250648206535158                Page ID2:   0.0759851790523549
    Page ID4:   0.182366091522817                Page ID1:   0.0348065095103204
-------------------------------------------------
    {1: [2, 4], 2: [3], 3: [4], 4: [5, 1], 5: [6], 6: []}
(After) PageRank of Graph 1: (Sorted by the PageRank value)
    Page ID4:   0.250381767330364                Page ID5:   0.156432222191375
    Page ID6:   0.187976776461659                Page ID3:   0.139910996936251
    Page ID1:   0.156432222191375                Page ID2:   0.108866014888975
```

2. Graph 2

```
    {1: [2], 2: [3], 3: [4], 4: [5], 5: [1]}
(Before) Graph 2
>> Authority:                                  >> Hub:
    1:  0.2                                      1:  0.2
    2:  0.2                                      2:  0.2
    3:  0.2                                      3:  0.2
    4:  0.2                                      4:  0.2
    5:  0.2                                      5:  0.2
-------------------------------------------------
    {1: [2, 4], 2: [3], 3: [4], 4: [5, 1], 5: [1]}
(After) Graph 2
>> Authority:                                  >> Hub:
    1:  0.309008271462044                        1:  0.309008271462044
    4:  0.309008271462044                        4:  0.309008271462044
    2:  0.190977613415013                        3:  0.190977613415013
    5:  0.190977613415013                        5:  0.190977613415013
    3:  0.0000282302458854417                    2:  0.0000282302458854417
```

```
================================================
    {1: [2], 2: [3], 3: [4], 4: [5], 5: [1]}
(Before) PageRank of Graph 2: (Sorted by the PageRank value)
    Page ID2:   0.200000000000000             Page ID1:   0.2
    Page ID3:   0.200000000000000             Page ID4:   0.2
    Page ID5:   0.200000000000000
------------------------------------------------
    {1: [2, 4], 2: [3], 3: [4], 4: [5, 1], 5: [1]}
(After) PageRank of Graph 2: (Sorted by the PageRank value)
    Page ID4:   0.277549253627240             Page ID5:   0.147980329786546
    Page ID1:   0.273734877973394             Page ID2:   0.146346941444894
    Page ID3:   0.154388597167926
```

3. Graph 3

```
    {1: [2], 2: [1, 3], 3: [2, 4], 4: [3]}
(Before) Graph 3
>> Authority:                              >> Hub:
    2:  0.309018567639257                     2:  0.309018567639257
    3:  0.309018567639257                     3:  0.309018567639257
    1:  0.190981432360743                     1:  0.190981432360743
    4:  0.190981432360743                     4:  0.190981432360743
------------------------------------------------
    {1: [2, 4], 2: [1, 3], 3: [2, 4], 4: [3, 1]}
(After) Graph 3
>> Authority:                              >> Hub:
    1:  0.25                                  1:  0.25
    2:  0.25                                  2:  0.25
    3:  0.25                                  3:  0.25
    4:  0.25                                  4:  0.25
================================================
    {1: [2], 2: [1, 3], 3: [2, 4], 4: [3]}
(Before) PageRank of Graph 3: (Sorted by the PageRank value)
    Page ID2:   0.324547068321367             Page ID1:   0.175452931678633
    Page ID3:   0.324547068321367             Page ID4:   0.175452931678633
------------------------------------------------
    {1: [2, 4], 2: [1, 3], 3: [2, 4], 4: [3, 1]}
(After) PageRank of Graph 3: (Sorted by the PageRank value)
    Page ID1:   0.25                          Page ID3:   0.25
    Page ID2:   0.25                          Page ID4:   0.25
```

- **討論：**

1. Graph 1:

Before:

- 因為沒有任何頁面指向Node 1，因此Authority(Node 1) = 0, PageRank也排名最低。
- 因為除了Node 6外，每個Node都指向1個Node，因此Hub(Node 1) = 0.2

After:

- 因為Node 4指向 Node 1，因此Authority(Node 1) = 0.2, PageRank也提高了。
- 因為Node 1多指向1個node(Node 4)，因此Hub值提高為 0.618。

2. Graph 2:

   Before:

   - 因為links是一個circle，因此所有Node的Authority及Hub皆 = 0.2, PageRank也皆 = 0.2。

   After:

   - 因為Node 4多指向 Node 1、Node 1多指向 Node 4，因此 Node 1 和 Node 4 的 Authority 及 Hub 值一起提高為 0.309，PageRank排名也一起提升。

3. Graph 3:

   Before:

   - 因為links是 1⇔2⇔3⇔4，因此邊緣Node(Node 1 & Node 4)的Authority 及 Hub 皆較低(= 0.191), PageRank也較低(= 0.175)。

   After:

   - 因為將Node 4指向 Node 1、Node 1指向 Node 4，使邊緣Node不再邊緣，Graph 3變為一個circle，因此Node 1 和Node 4的Authority及Hub值一起提高為 0.25。PageRank排名也一起提升(所有 Node 的 PageRank 皆 = 0.25)。

4. 加入{‘1’:[4], ‘4’:[1]}是經過幾次測試後最簡單、通用於3個Graph的added_links。一開始想要增加{'1': [3, 6], '4': [1], '6': [1]}，然而發現Graph 2, 3沒有Node 6，因此想到可以用「加入{‘1’:[4], ‘4’:[1]}」的方式就達到增加 Node 1 的Authority、Hub、PageRank值的方法！

- **綜合討論：**

1. 利用Project 3的機會，清楚認識了HITS、PageRank及SimRank的實作方式及差異。

2. 然而這三個演算法似乎都沒有特別針對網頁本身的quality（對於user的幫助、提供的資料是否重要/有用？）做評分，不過如何對這部分做評分，可能也是另外一個issue。

3. 在實際上處理時，可能也需要考慮子網站等內部link的狀況。

4. 現在有許多網站提供查詢webpage的rank，例如Alexa(www.alexa.com/siteinfo/ by Amazon)，提供了Traffic Ranks、分析該網站造訪者來自哪些國家等地域分析。