Data Mining hw3

資工碩一 P76074240 蔡文傑

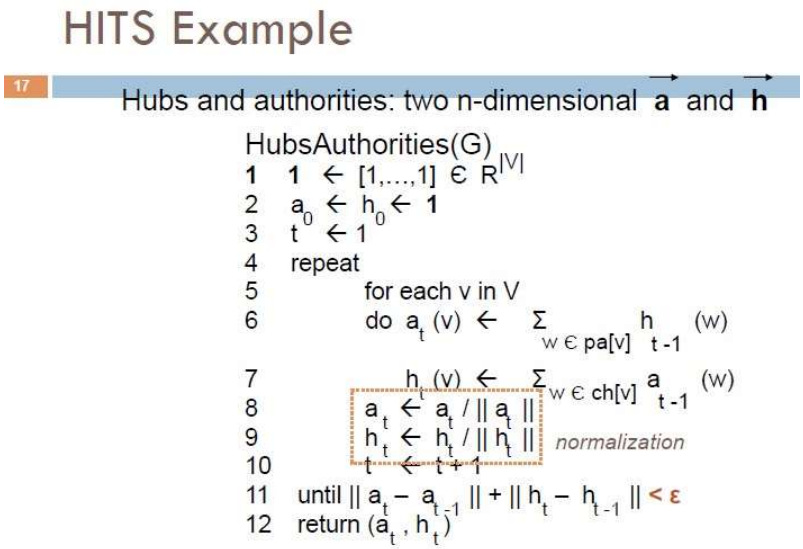
1. Implementation detail

以下包含 HITS、PageRank 與 SimRank 演算法實作程式碼，每一個演算法皆為一個函示，完整內容於 hw3_code.py。

本次作業採用一個 jupyter notebook 呼叫 hw3_code.py 執行，執行過程與結果請見 hw3_code_execute_result.ipynb，或是 HTML 網頁格式版本的 hw3_code_execute_result.html。

1.1 HITS

依據講義 7_Link Analysis.pdf 第 17 頁演算法虛擬碼進行實作，




```python
def Hits(G, theta=1e-5, print_interation_count=False):
    setG = set(np.asarray(G).flatten())
    nvertex = len(setG)
    auth = np.ones(nvertex+1) # authoritative
    hub = np.ones(nvertex+1) # hub

    # graph relation
    pa = [[] for _ in range(nvertex+1)] # parient
    ch = [[] for _ in range(nvertex+1)] # children
    for m, n in G:
        # m -> n
        pa[n].append(m)
```

```
            ch[m].append(n)

    # power on iteration
    t = 1
    while True:
        # print('iter: ', t)
        new_auth = np.zeros(nvertex+1)
        new_hub = np.zeros(nvertex+1)
        for v in setG: # for each node in graph
            for w in pa[v]:
                new_auth[v] += hub[w]

            for w in ch[v]:
                new_hub[v] += auth[w]

        # normalization
        new_auth /= np.sum(new_auth[1:])
        new_hub /= np.sum(new_hub[1:])

        # check converage
        diff = (np.sum(np.abs(new_auth[1:] - auth[1:])) +
np.sum(np.abs(new_hub[1:] - hub[1:])))
        # print('diff: ', diff)
        if (diff < theta): break
        auth = new_auth
        hub = new_hub
        t += 1

    if print_interation_count:
        print('converage at {} iterations.'.format(t))

    return (new_auth[1:], new_hub[1:])
```

auth 與 hub 分別代表 authoritative 與 hub。首先，程式會先建立一個 pa[]與 ch[]儲存圖之間的關係，pa[]指向節點的 parient，ch[]指向節點 children。接著逐次依照演算法逐次迭代計算新的 new_auth 與 new_hub，每一次迭代會計算與前一次差異 diff，如果差異小於定義的 theta 則跳出迴圈並回傳計算結果。

1.2 PageRank

依據講義 7_Link Analysis.pdf 第 37 頁公式實作。

Quick reference

37

$$PR(P_i) = \frac{(d)}{n} + (1-d) \times \sum_{l_{j,i} \subset E} PR(P_j) / \text{Outdegree}(P_j)$$

D(damping factor)=0.1~0.15
n=|page set|

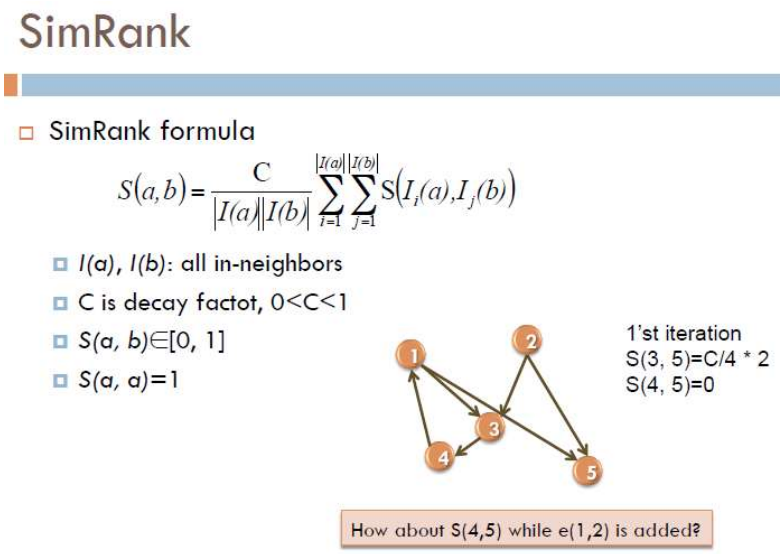
```
def PageRank(G, theta =1e-5, damping_factor=0.15,
print_interation_count=False):
    setG = set(np.asarray(G).flatten())
    nvertex = len(setG)

    # Page Rank initial value
    # set vertex 1 with index 1
    PR = np.ones(nvertex+1) / (nvertex)

    # build graph relation
    pa = [[] for _ in range(nvertex+1)] # parient
    outdeg = np.zeros(nvertex+1) # out degree
    for m, n in G:
        # m -> n
        pa[n].append(m)
        outdeg[m] += 1

    t = 1
    while True:
        # print('iter: ', t)
        # power on interation
        new_PR = np.zeros(nvertex+1)
        for node in setG: # for each node in graph
            # print('node: ', node)
            for p in pa[node]: # find the parient of this node
                # print(p)
                new_PR[node] += PR[p]/outdeg[p]

            # damping factor
            new_PR[node] = (damping_factor/nvertex) + (1-
damping_factor)*new_PR[node]

        # print('PR: ', new_PR[1:])
        diff = np.sum(np.abs(new_PR[1:] - PR[1:])) # calculate the
difference with previous iteration
        # print('diff: ', diff)
        if (diff < theta): break
        PR = new_PR
```

```
        t += 1

    if print_interation_count:
        print('converage at {} iterations.'.format(t))

    return new_PR[1:]
```

與前面 HITS 一樣採用迭代計算，不過這次採用 PR 陣列儲存 PageRank 計算結果。每一次迭代 new_PR 會依據講義上公式計算出新的 PageRank，然後經過 damping factor 處理，最後計算與前一輪差異並判斷是否小於定義的 theta 來結束迭代。

1.3 SimRank

依據講義 7_Link Analysis.pdf 第 52 頁公式實作。




```
def SimRank(G, C=0.5, theta=1e-5, print_interation_count=True):
    setG = set(np.asarray(G).flatten())
    nvertex = len(setG)

    # SimRank init value
    simrank = np.zeros((nvertex+1, nvertex+1))

    # build graph relation
    inlink = [[] for _ in range(nvertex+1)]
    for m, n in G:
        # m --> n
        inlink[n].append(m)

    # power on iteration
    t = 1
```

```
    while True:
        new_simrank = np.zeros((nvertex+1, nvertex+1))
        for a in range(1, nvertex+1):
            for b in range(1, nvertex+1):
                # print(a, b)

                # simrank(a,a)=1
                if a == b:
                    new_simrank[a][b] = 1
                    continue

                # count inlink number
                inlinkNum_a = len(inlink[a])
                inlinkNum_b = len(inlink[b])
                if inlinkNum_a==0 or inlinkNum_b==0: continue

                tmp = 0
                for i in inlink[a]:
                    for j in inlink[b]:
                        if i == j:
                            tmp += 1
                        else:
                            tmp += simrank[i, j]

                new_simrank[a][b] = C/(inlinkNum_a*inlinkNum_b)*tmp

        diff = np.sum(np.abs(new_simrank[1:, 1:] - simrank[1:, 1:]))
# l1 norm
        if diff < theta: break # check converage
        simrank = new_simrank # if not converage, keep iter
        t += 1

    if print_interation_count:
        print('converage at {} iterations.'.format(t))

    return(new_simrank[1:, 1:])
```

雖然在公式採用遞迴定義，不過在實作時採用的是迭代的方式，一開始定義 simrank[i][j]二維陣列，表示 simrank(i, j)的值，接著程式會會依據公式不斷進行迭代更新新的 new_simrank 值，每次迭代最後都會檢查與前一回的 simrank 差異來判斷是否收斂。

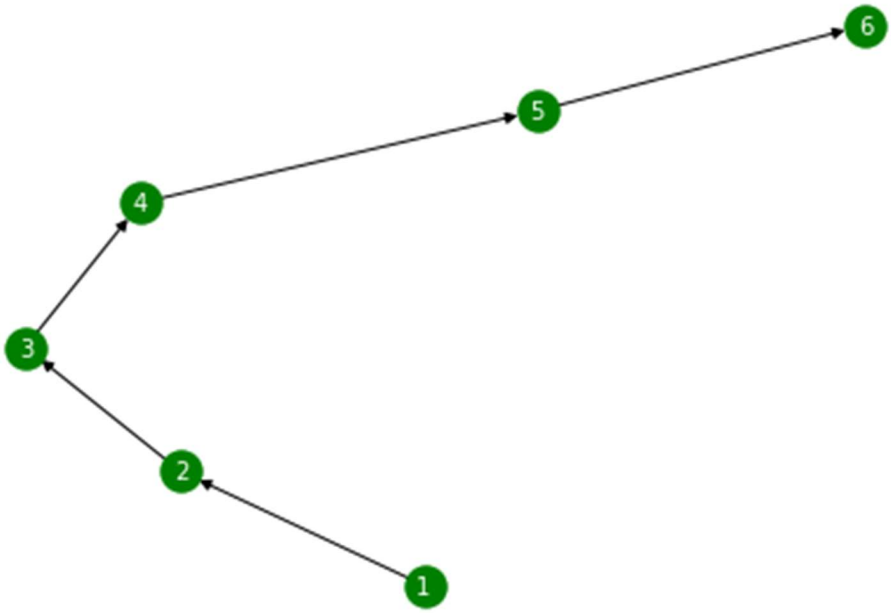
2. Result analysis and discussion

本次作業將會執行以下 8 個檔案。其中編號 1~6 為 hw3 作業提供的資料集，編號 7~8 為 hw1 的 transation 資料經過轉換而成的單向圖與雙向圖，轉換過程於 convert_transation_data_to_graph.html。

編號	檔案名稱	備註
1	`'./hw3dataset/graph_1.txt',`	hw3 題目提供資料
2	`'./hw3dataset/graph_2.txt',`	
3	`'./hw3dataset/graph_3.txt',`	
4	`'./hw3dataset/graph_4.txt',`	
5	`'./hw3dataset/graph_5.txt',`	
6	`'./hw3dataset/graph_6.txt',`	
7	`'./hw1_trasation_dataset/hw1_transdata_digraph.csv',`	hw1 轉換的單向圖
8	`'./hw1_trasation_dataset/hw1_transation_bigraph.csv'`	hw1 轉換的雙向圖

在 hw3_code_execute_result.html 包含了以上資料集的執行結過，接下來會針對這些資料集做討論。在此只會列出需要被討論的值，其他完整 hits, pages, simrank 請見上述的網頁檔案。

2.1 編號 1 graph_1.txt



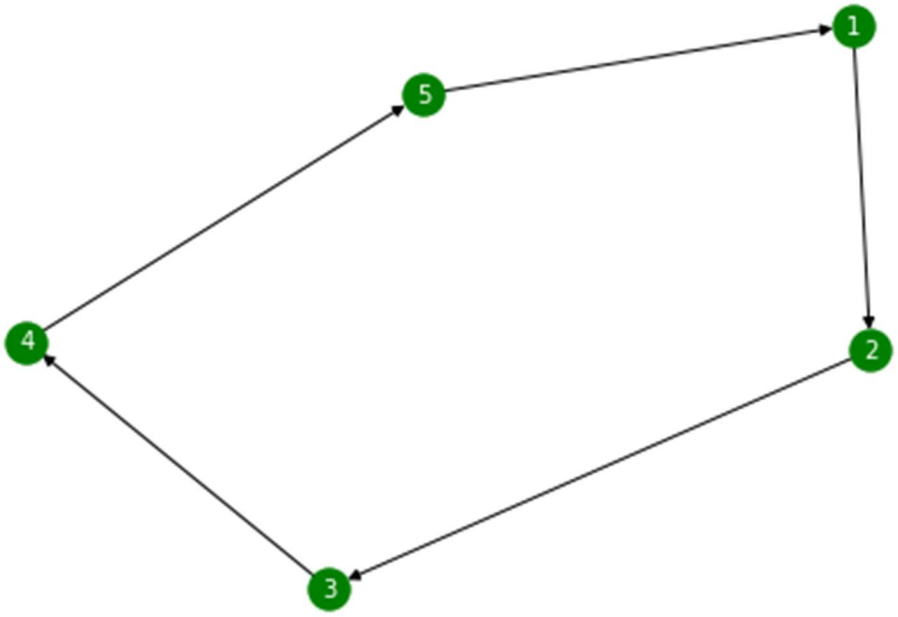
```
Hits:
converage at 2 iterations.
wall time:  0.00020253704860806465
node 1: auth=0.0, hub=0.2
node 2: auth=0.2, hub=0.2
node 3: auth=0.2, hub=0.2
node 4: auth=0.2, hub=0.2
node 5: auth=0.2, hub=0.2
node 6: auth=0.2, hub=0.0
```

由圖的分布可觀察到呈現一直線分布，其中節點 2 3 4 5 都有一條 inlink 與 outlink 因此這些點的 authority, hub 值皆相等，而整條線的起點 1 因為有指向別的節點但是沒有被其他節點指到所以 auth=0.0, hub=0.2，而最末端節點 6 則剛好相反 auth=0.2, hub=0.0。

```
PageRank:
converage at 7 iterations.
wall time:  0.00011163600720465183
node 1: PageRank=0.024999999999999998
node 2: PageRank=0.04625
node 3: PageRank=0.0643125
node 4: PageRank=0.07966562499999999
node 5: PageRank=0.09271578124999999
node 6: PageRank=0.10380841406249998
```

從 pagerank 可觀察到節點會受到其父節點的分數影響，因此最後被指向的節點 6 有最高的 pagerank 然後依序遞減到節點 1

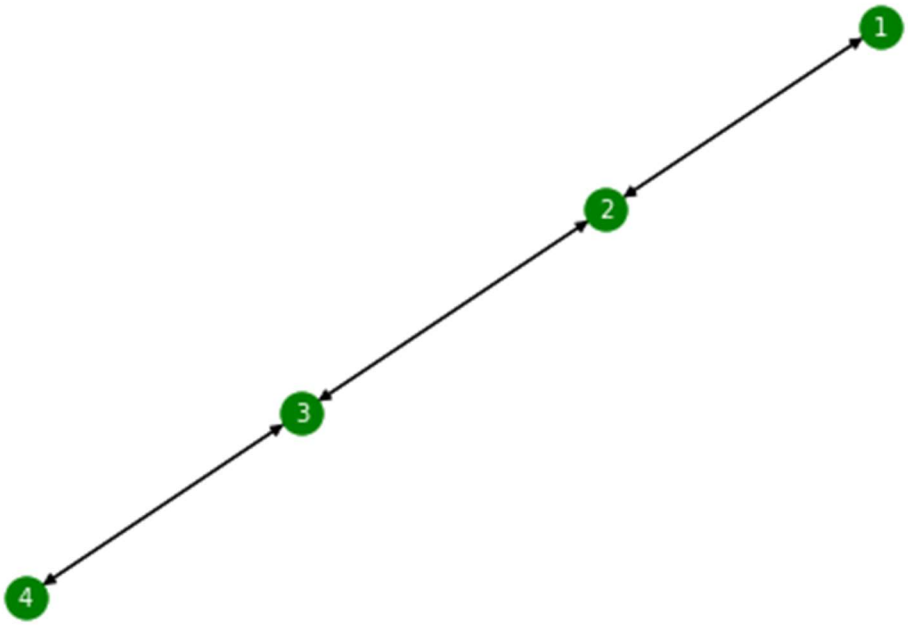
2.2 編號 2 graph_2.txt




```
Hits:
converage at 2 iterations.
wall time:  0.00012256496120244265
node 1: auth=0.2, hub=0.2
node 2: auth=0.2, hub=0.2
node 3: auth=0.2, hub=0.2
node 4: auth=0.2, hub=0.2
node 5: auth=0.2, hub=0.2


PageRank:
converage at 1 iterations.
wall time:  6.078003207221627e-05
node 1: PageRank=0.2
node 2: PageRank=0.2
node 3: PageRank=0.2
node 4: PageRank=0.2
node 5: PageRank=0.2
```

此圖中的節點形成一個圓圈，因此每一個節點的 authority, hub, PageRank 值皆相等。

2.3 編號 3 graph_3.txt




```
Hits:
converage at 13 iterations.
wall time:  0.0003507339861243963
node 1: auth=0.1909827760891591, hub=0.1909827760891591
node 2: auth=0.3090172239108409, hub=0.3090172239108409
node 3: auth=0.3090172239108409, hub=0.3090172239108409
node 4: auth=0.1909827760891591, hub=0.1909827760891591
================================================


PageRank:
converage at 14 iterations.
wall time:  0.00016009900718927383
node 1: PageRank=0.17543906418231678
node 2: PageRank=0.324560935817683
node 3: PageRank=0.324560935817683
node 4: PageRank=0.17543906418231678
================================================
```

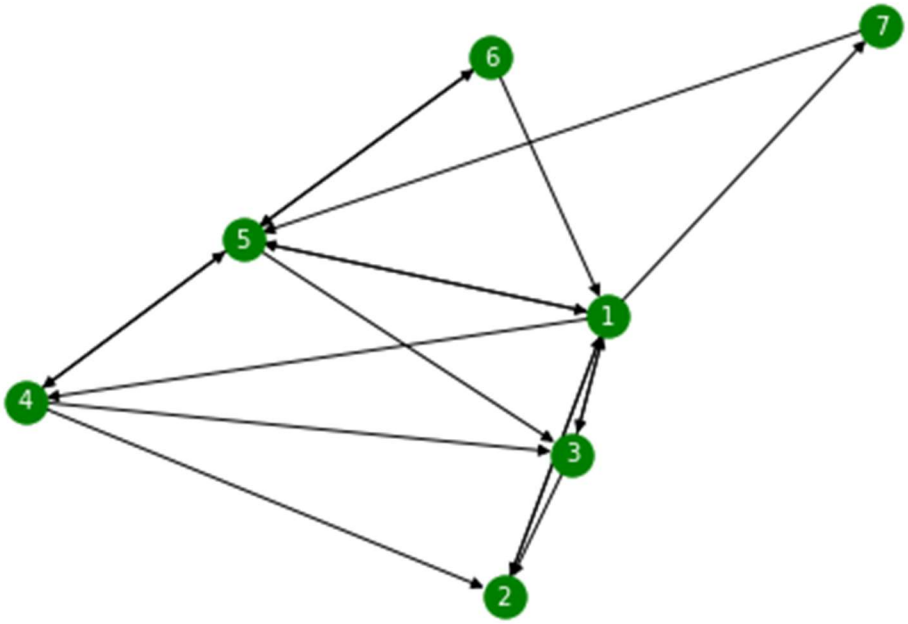
圖中可發現節點 2 3 不論 inlink 與 outlink 皆比較多，因此這兩個節點的 auth, hub 與 pagerank 值都有比較高的分數。

```
SimRank:
converage at 10 iterations.
```

```
wall time:  0.0002581229782663286
SimRank Matrix:
 [[1.         0.         0.33333302 0.        ]
 [0.         1.         0.         0.33333302]
 [0.33333302 0.         1.         0.        ]
 [0.         0.33333302 0.         1.        ]]

non-zero simrank:
simrank(1, 1) = 1.0
simrank(1, 3) = 0.33333301544189453
simrank(2, 2) = 1.0
simrank(2, 4) = 0.33333301544189453
simrank(3, 1) = 0.33333301544189453
simrank(3, 3) = 1.0
simrank(4, 2) = 0.33333301544189453
simrank(4, 4) = 1.0
================================================
```

另外，從 simrank 可以觀察到有對稱的特性，每間格一個節點的兩個節點之間的 simrank 值皆為 0.33333301544189453。

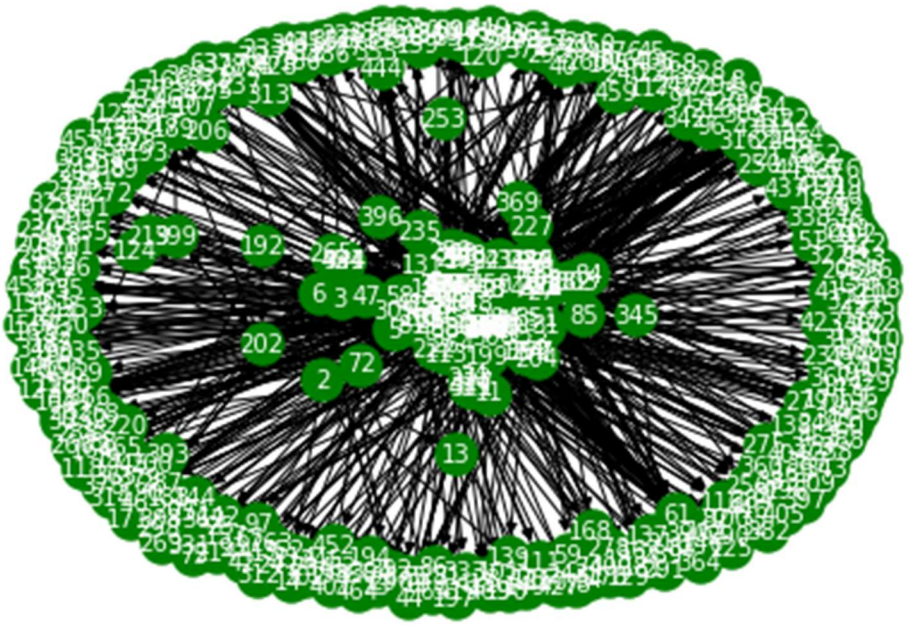
2.4 編號 4 graph_4.txt

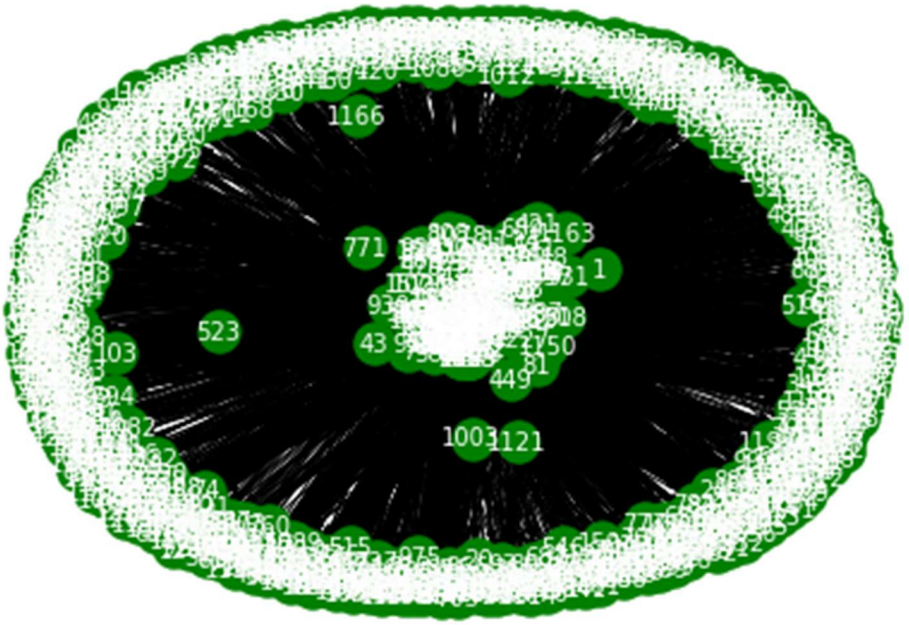


```
Hits:
converage at 22 iterations.
wall time:  0.0006535249995067716
node 1: auth=0.13948516705262493, hub=0.27545126233700823
node 2: auth=0.17791153263811105, hub=0.047763613216291544
node 3: auth=0.200822967744709, hub=0.1086840307492033
node 4: auth=0.14017785831704038, hub=0.1986580026583584
node 5: auth=0.20142461593047428, hub=0.1837361651233888
node 6: auth=0.056089763410282885, hub=0.1167352695660206
node 7: auth=0.0840880949067575, hub=0.06897165634972904
================================================


PageRank:
converage at 15 iterations.
wall time:  0.00025246996665373445
node 1: PageRank=0.2802877573557533
node 2: PageRank=0.15876468724856077
node 3: PageRank=0.13888170080462583
node 4: PageRank=0.10821930792439602
node 5: PageRank=0.18419866731369647
node 6: PageRank=0.06057050475273925
node 7: PageRank=0.0690773746002282
================================================
```

此圖相較於前面幾張較為複雜，不過仍然可觀察到 inlink 與 outlink 比較多的節點無論是在 HITS 與 Pagerank 皆會有比較高的分數。

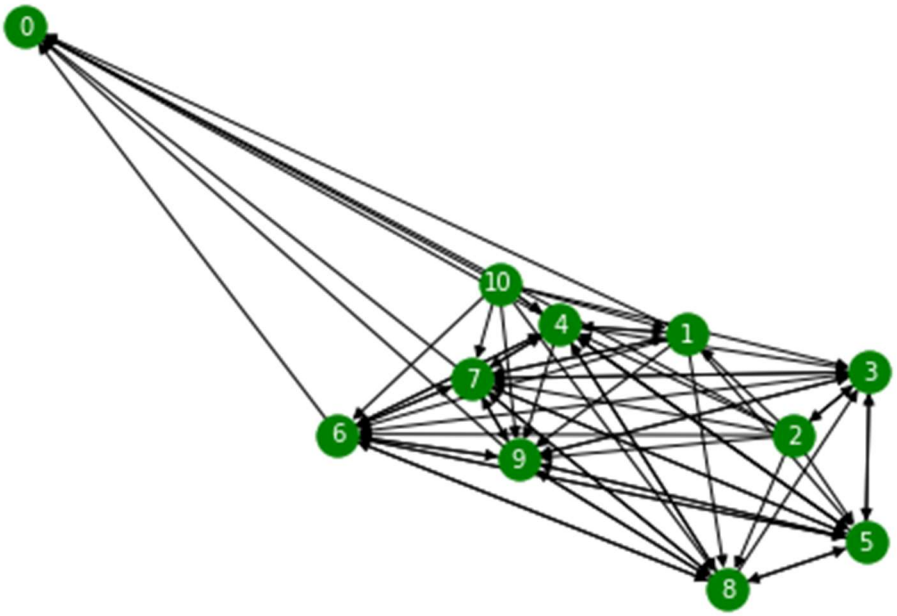
2.5 編號 5 graph_5.txt 與 編號 6 graph_6.txt

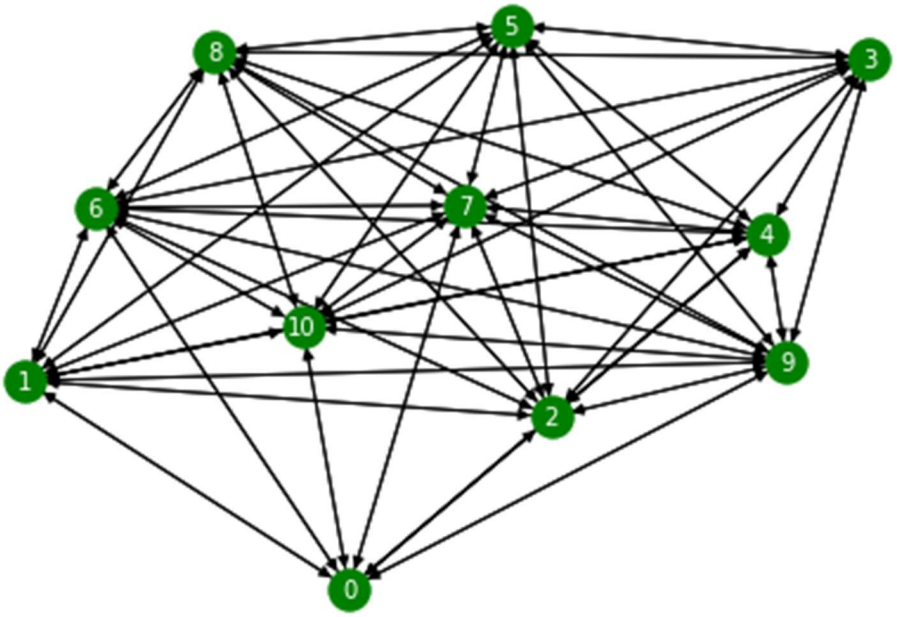




這兩張圖大致上為一顆分支度很高的樹，中間的節點會有比較高的 authority 而周圍的分支節點則會有較高的 hub 值。

2.6 編號 7 hw1_transdata_digraph.csv 與 編號 8 hw1_transation_bigraph.csv





在從原本單向圖變成雙向圖時，節點 7 的 inlink 與 outlink 值變多的，因次可觀察到原本最高的 pagerank 節點從原本節點 5 6 變為節點 7。不過大致上 HIT 與 PageRank 的分布的相對關係並沒有太大變動。

3. Computation performance analysis

接下來會針對執行時間進行探討。

HITS

編號	檔案名稱	執行時間	收斂迭代次數
1	`'./hw3dataset/graph_1.txt',`	0.00020253704860806465	2
2	`'./hw3dataset/graph_2.txt',`	0.00012256496120244265	2
3	`'./hw3dataset/graph_3.txt',`	0.00035073398612439 63	13
4	`'./hw3dataset/graph_4.txt',`	0.00065352499950 67716	22
5	`'./hw3dataset/graph_5.txt',`	0.01436753501184 2847	22
6	`'./hw3dataset/graph_6.txt',`	0.32640659401658 92	120
7	`'./hw1_trasation_dataset/hw1_transdata_digraph.csv',`	0.00061473599635 0646	8
8	`'./hw1_trasation_dataset/hw1_transation_bigraph.csv'`	0.00086393096717 07451	7

PageRank

編號	檔案名稱	執行時間	收斂迭代次數
1	`'./hw3dataset/graph_1.txt',`	0.00011163600720465183	7

2	`'./hw3dataset/graph_2.txt',`	6.0780032072216 27e-05	1
3	`'./hw3dataset/graph_3.txt',`	0.0001600990071 8927383	14
4	`'./hw3dataset/graph_4.txt',`	0.0002524699666 5373445	15
5	`'./hw3dataset/graph_5.txt',`	0.0081280020531 26693	13
6	`'./hw3dataset/graph_6.txt',`	0.0321907170000 4861	11
7	`'./hw1_trasation_dataset/hw 1_transdata_digraph.csv',`	0.0012743789702 653885	37
8	`'./hw1_trasation_dataset/hw 1_transation_bigraph.csv'`	0.0006205089739 523828	7

SimRank

編號	檔案名稱	執行時間	收斂迭代次數
1	`'./hw3dataset/graph_1.txt',`	0.0001053679734 4684601	2
2	`'./hw3dataset/graph_2.txt',`	0.0001094000181 183219	2
3	`'./hw3dataset/graph_3.txt',`	0.0002581229782 663286	10
4	`'./hw3dataset/graph_4.txt',`	0.0014333540457 300842	16
5	`'./hw3dataset/graph_5.txt',`	8.5281302019720 9	24
6	`'./hw3dataset/graph_6.txt',`	x	x
7	`'./hw1_trasation_dataset/hw 1_transdata_digraph.csv',`	x	x
8	`'./hw1_trasation_dataset/hw 1_transation_bigraph.csv'`	x	x

由上面基本上可觀察到執行時間與收斂迭代次數會隨著圖的大小而增加。在編號 7 與編號 8 的執行結果可發現當節點數固定時，雙向圖的收斂速度會比單向圖還要快。

4. Questions & Discussion

1 PageRank 事實上還有另外一種透過矩陣相乘的方式進行運算，不過考量當圖中節點非常多的時候此矩陣會變得非常大，會增加計算成本，因此本次作業採用的是透過一個陣列只儲存在圖中有邊的節點，如此可減少儲存記憶體消耗與加快執行速度。

2 最初在實作 simrank 演算法的時候，我採用的是遞迴的方式並設定一個最大遞迴深度的方式，可是後來在測試時發現即使只要節點稍微變多，就會需要非常多的計算時間，因此最後改用迭代的方式計算。

3 Can link analysis algorithms really find the “important” pages from Web?

首先，必須要先定義什麼是重要的網頁。一個網頁的重要性可以包含 Intra-page 資訊 inter-page 資訊，Intra-page 資訊在網路世界來說就是連結，透過連結分析演算法如 HITS、Pagerank 就可以做到，但是要做到 inter-page 資訊的了解卻不是那麼容易的一件事，目前還有沒一種單一演算法可以直接得出網頁 inter-page 資訊到底哪個比較好。

4 What are practical issues when implement these algorithms in a real Web?

4.1 回朔找到網頁的父節點非常困難且成本很高。

4.2 網頁可能會有沒有連結，或是指向自己的連結的異常情況。

4.3 真實網路中網頁之間的連結隨時都在變，可能當今天計算完 pagerank 分數之後，隔天這個分數就不適用了。

5 What is the effect of “C” parameter in SimRank?

因為 simrank 是一個遞迴定義的公式，可能會發生無限遞迴情況。因此在每一輪計算時如果都乘以一個 0~1 的值，遞迴到最後就會變成趨近於零的值。在實作時只要偵測這個回傳值是否小於一個閥值(如:0.001)就可以跳離無限遞迴並回傳零。