Data Mining 資料探勘 Project3 書面報告

- **Implementation detail**

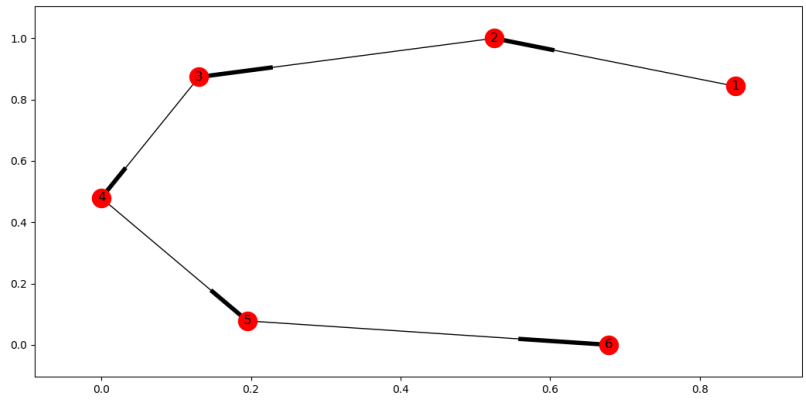
本次 code 使用 python 函式庫 networkx 做 input graph，自行寫 hits, pagerank 及 simrank 的運算，此程式需要裝 networkx(pip3 install networkx)

```
# First step, create graph in order to perform hits and pagerank
def build_graph(path):
    graph_data = pd.read_csv(path,header=None)
    graph_data = graph_data.values.tolist()
    G = nx.DiGraph()
    G.add_edges_from(graph_data)
    return G
```

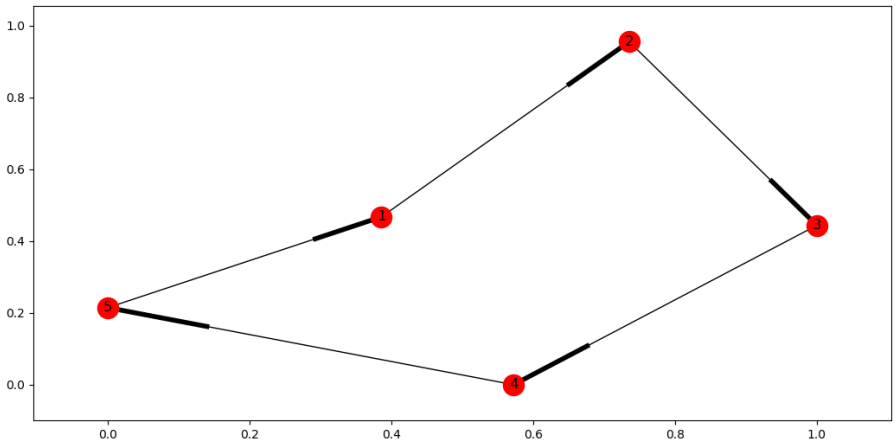
此部分利用 pandas 讀入 csv，務必記得將 header 設置為 None，讀出的格式為 dataframe，故使用 values.tolist()轉為 list，才可放入 nx 建成 graph，這裡使用 DiGraph()建置出來的是 Directed Graph，如果要建置 undirected Graph 要使用 Graph()，利用 add_edges_from 可以將 list 內的關係加到圖內。

```
# Used to plot graph
def graph_plot(G):
    plt.figure(figsize =(10, 10))
    nx.draw_networkx(G, with_labels = True)
    plt.show()
```

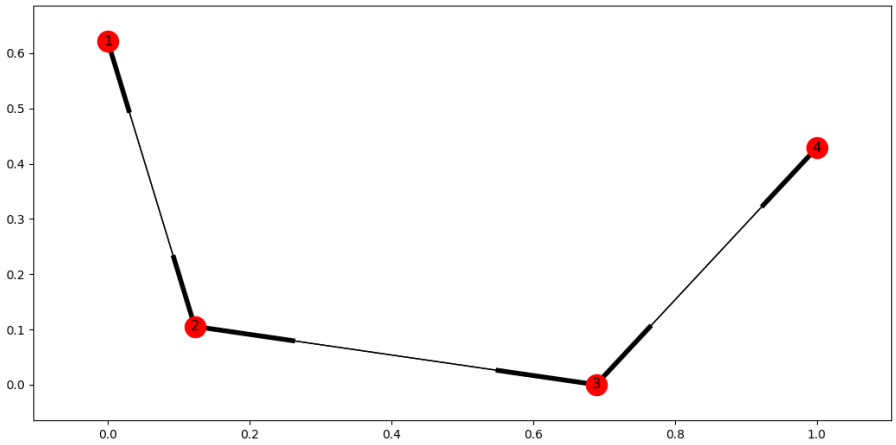
此部分用在繪製出 graph object 的樣子，可以搭配 matplotlib 直接繪製出 directed graph，因 graph_5 及 graph_6 和 transection_data 的 node 數量太多，所以畫出來會整個雜亂無法辨識，所以這邊指呈現出 graph1~4 的樣子。



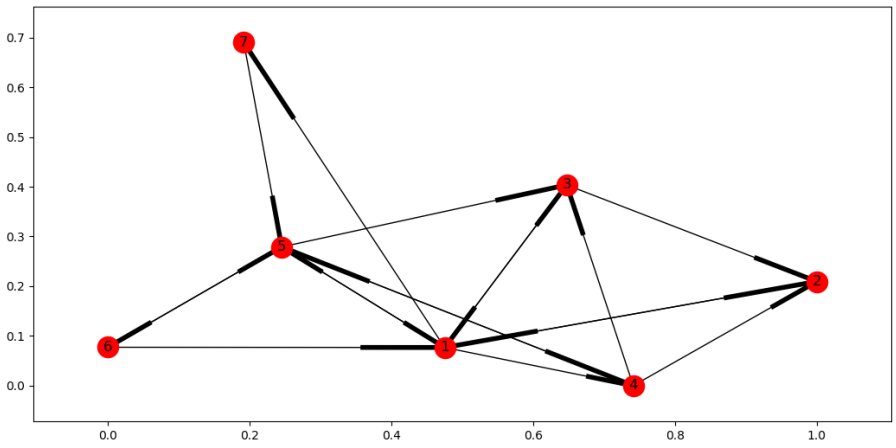
Graph 1



Graph 2



Graph 3



Graph 4

```
def hitsandpr(G, damping_factors=0.15):
    startTime = time()
    hubs, authorities = hits(G)
    print("HITS computation time ::", time() - startTime)
    #print("Hub Scores: ", hubs)
    #print("Authority Scores: ", authorities)
    startTime = time()
    pr = pagerank(G, damping_factors, max_iter=500)
    print("PageRank computation time ::", time() - startTime)
    #print("PageRank Values : ", pr)
    return hubs, authorities, pr
```

```
def hits(graph, iter_count=500):
    nodes = graph.nodes()
    nodes_count = len(nodes)
    matrix = nx.to_numpy_matrix(graph, nodelist=nodes)

    hubs_score = np.ones(nodes_count)
    auth_score = np.ones(nodes_count)
    H = matrix * matrix.T
    A = matrix.T * matrix

    for i in range(iter_count):

        hubs_score = hubs_score * H
        auth_score = auth_score * A
        hubs_score = hubs_score / LA.norm(hubs_score)
        auth_score = auth_score / LA.norm(auth_score)

    hubs_score = np.array(hubs_score).reshape(-1,)
    auth_score = np.array(auth_score).reshape(-1,)

    hubs = dict(zip(nodes, hubs_score/2))
    authorities = dict(zip(nodes, auth_score/2))
    return hubs, authorities
```

```
def pagerank(graph, d, max_iter=500):
    W = nx.stochastic_graph(graph, weight='weight')
    N = W.number_of_nodes()
    x = dict.fromkeys(W, 1.0 / N)
    p = dict.fromkeys(W, 1.0 / N)

    for _ in range(max_iter):
        xlast = x
        # {1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0}
        x = dict.fromkeys(xlast.keys(), 0)
        for n in x:
            for nbr in W[n]:
                x[nbr] += (1-d) * xlast[n] * W[n][nbr]['weight']
            x[n] += d * p[n]

    pr_sorted = sorted(
        x.items(), key=lambda v: (v[1], v[0]), reverse=True)
    print("\n\nThe order generated by d library is")
    for i in pr_sorted:
        print(i[0], end=" ")
    print()
```

此部分的 code 用於實現 hits 和 pagerank 演算法，用 hitsandpr 去呼叫 hits 可算出 hubs 和 authorities，這個部分順便使用 time 來記錄運算過程所花的

時間，pagerank 可以設定 damping_factors，結果分析部分會針對這塊多做實驗。

```
def simrank(G, r=0.8, max_iter=100, eps=1e-4):

    nodes = G.nodes()
    nodes_i = {k: v for(k, v) in [(nodes[i], i) for i in range(0, len(nodes))]}

    sim_prev = np.zeros(len(nodes))
    sim = np.identity(len(nodes))

    for i in range(max_iter):
        if np.allclose(sim, sim_prev, atol=eps):
            break
        sim_prev = np.copy(sim)
        for u, v in itertools.product(nodes, nodes):
            if u is v:
                continue
            u_ns, v_ns = G.predecessors(u), G.predecessors(v)

            # evaluating the similarity of current iteration nodes pair
            if len(u_ns) == 0 or len(v_ns) == 0:
                # if a node has no predecessors then setting similarity to zero
                sim[nodes_i[u]][nodes_i[v]] = 0
            else:
                s_uv = sum([sim_prev[nodes_i[u_n]][nodes_i[v_n]] for u_n, v_n in itertools.product(u_ns, v_ns)])
                sim[nodes_i[u]][nodes_i[v]] = (r * s_uv) / (len(u_ns) * len(v_ns))


    return sim
```

上圖的 r 即為 Decay_Factor C

這一個 part 實現了 simrank，最後會產出一個節點間相關的 value 的矩陣，如下圖：

	A	B	C	D	E	F
1	1	0	0	0	0	0
2	0	1	0	0	0	0
3	0	0	1	0	0	0
4	0	0	0	1	0	0
5	0	0	0	0	1	0
6	0	0	0	0	0	1

目前分成三個主程式，實作 HITS.py 和 Pagerank.py 和 simrank 資料夾中的 simrank.py，另外 increase.py 用於實驗 pagerank 中不同 damping_factor 的狀況。hits_and_pagerank_graph1to6.py 則是對六張圖做增加 hub, authority, and PageRank 的分析用。

➢ **Result analysis and discussion**

老師上課提到以下：

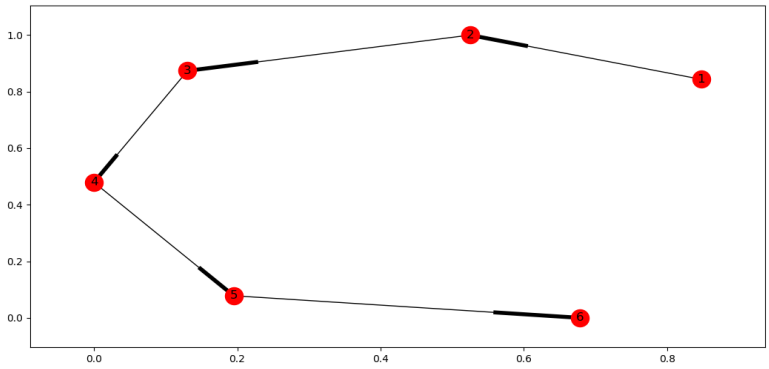
- ✧ A site is very **authoritative** if it **receives many citations**. Citation from **important sites weight more** than citations from less-important sites.
- ✧ Hubness shows the importance of a site. A good hub is a site that **links to many authoritative sites**.

將會根據上述兩點做分析。

第一部分，Hub & Authority：

Graph1:

Hub	Authority
0.2	0
0.2	0.2
0.2	0.2
0.2	0.2
0.2	0.2
0	0.2

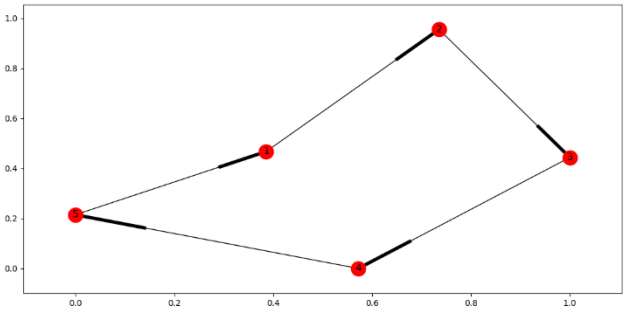


由此圖與表格可以歸納出：

- ✧ 有指向其他點的節點，有 Hub 值。
- ✧ 沒有被指向的節點，沒有 Authority 值。

Graph2:

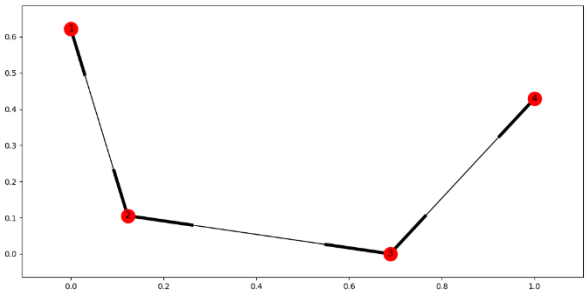
Hub	Authority
0.2	0.2
0.2	0.2
0.2	0.2
0.2	0.2
0.2	0.2



此圖重點為，每個節點都有指向且也有被指向，貢獻相同，Hub 與 Authority 相同。

Graph3:

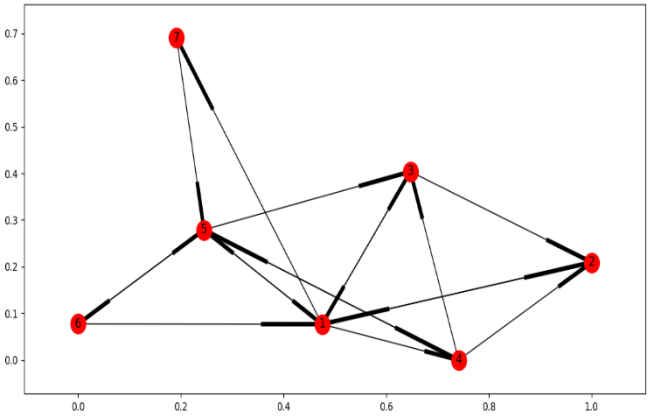
Hub	Authority
0.190983	0.190983
0.309017	0.309017
0.309017	0.309017
0.190983	0.190983



2,3 節點同樣為被兩個節點指向(authority 高)，且指向兩節點(其中節點 3 的 authority 較節點 1 高)的節點(hub 高因為連到較 authorative 的節點)。

Graph4:

Hub	Authority
0.275453	0.139484
0.047762	0.177912
0.108683	0.200823
0.19866	0.140178
0.183735	0.201425
0.068972	0.084088
0.116735	0.056089



從 hub 值最高的節點 1 開始分析，其分別指向節點 2,3,5 Authority 較高的節點，因此 hub 值較高，節點 5 有著相對重要的節點 1 指向，以節點 4,6,7 指向，authority 相對較高。

Graph5,Graph6,Transection dataset 因資料較多皆不再此贅述，但確實照著老師上課所說，從 hub 值與 authority 值的高低可以看出節點的重要性，若要參考詳細結果，有附上 csv 檔。

第二部分，Pagerank

- ✧ Pagerank 相較 HITS，其較重視網站被什麼樣的網站給指向，如果是較重要的網站「**單獨**」指向，pr 值會較高。

這個部分我將 damping_factor 設置為 0.15 做為初始值，產生以下的值。

節點	Graph_1	Graph_2	Graph_3	Graph_4
1	0.14596	0.2	0.232558	0.169027
2	0.167853	0.2	0.267442	0.143567
3	0.171138	0.2	0.267442	0.13919
4	0.17163	0.2	0.232558	0.132562
5	0.171704	0.2		0.161664
6	0.171715			0.126499
7				0.127491

- ✧ Graph1 可看出 pagerank 較重視被網站指向，沒有被任何節點指向的節點 1 相對 pr 值較低。
- ✧ 可以看出 Graph2 較特別，因每一個節點都有同樣的 inlink 和 outlink，parent 與 parent 間的相連關係更是相同，故皆有一樣的值。
- ✧ Graph3 的狀況與 HITS 較相同，節點 2 與 3 有相同重要節點的 inlink。
- ✧ Graph4 中節點 1 的 pr 值較高的關鍵，應該是節點 2 的影響，其有兩個 inlink 卻，但其 pr 值全部貢獻給 1，第二高的節點 5 也有同樣的情況，較低的原因應該是節點 7 相對節點 2 的 inlink 較少，其他節點幾乎都有兩個以上的 outlink。

Graph5,Graph6,Transection dataset 因資料較多皆不再此贅述，若要參考詳細結果，有附上 csv 檔。

第三部分，SimRank

✧ 在此部分，C 由於是自訂，在此先將 C 設為 0.8。

	1	2	3	4	5	6
1	1	0	0	0	0	0
2	0	1	0	0	0	0
3	0	0	1	0	0	0
4	0	0	0	1	0	0
5	0	0	0	0	1	0
6	0	0	0	0	0	1

Graph 1

	1	2	3	4	5
1	1	0	0	0	0
2	0	1	0	0	0
3	0	0	1	0	0
4	0	0	0	1	0
5	0	0	0	0	1

Graph 2

	1	2	3	4
1	1	0	0.666597	0
2	0	1	0	0.666597
3	0.666597	0	1	0
4	0	0.666597	0	1

Graph 3

	1	2	3	4	5	6	7
1	1	0.360139	0.348831	0.353604	0.337524	0.292245	0.414963
2	0.360139	1	0.406676	0.36962	0.412072	0.45395	0.28529
3	0.348831	0.406676	1	0.449462	0.389935	0.450935	0.447988
4	0.353604	0.36962	0.449462	1	0.342558	0.534985	0.534985
5	0.337524	0.412072	0.389935	0.342558	1	0.412126	0.272989
6	0.292245	0.45395	0.450935	0.534985	0.412126	1	0.269969
7	0.414963	0.28529	0.447988	0.534985	0.272989	0.269969	1

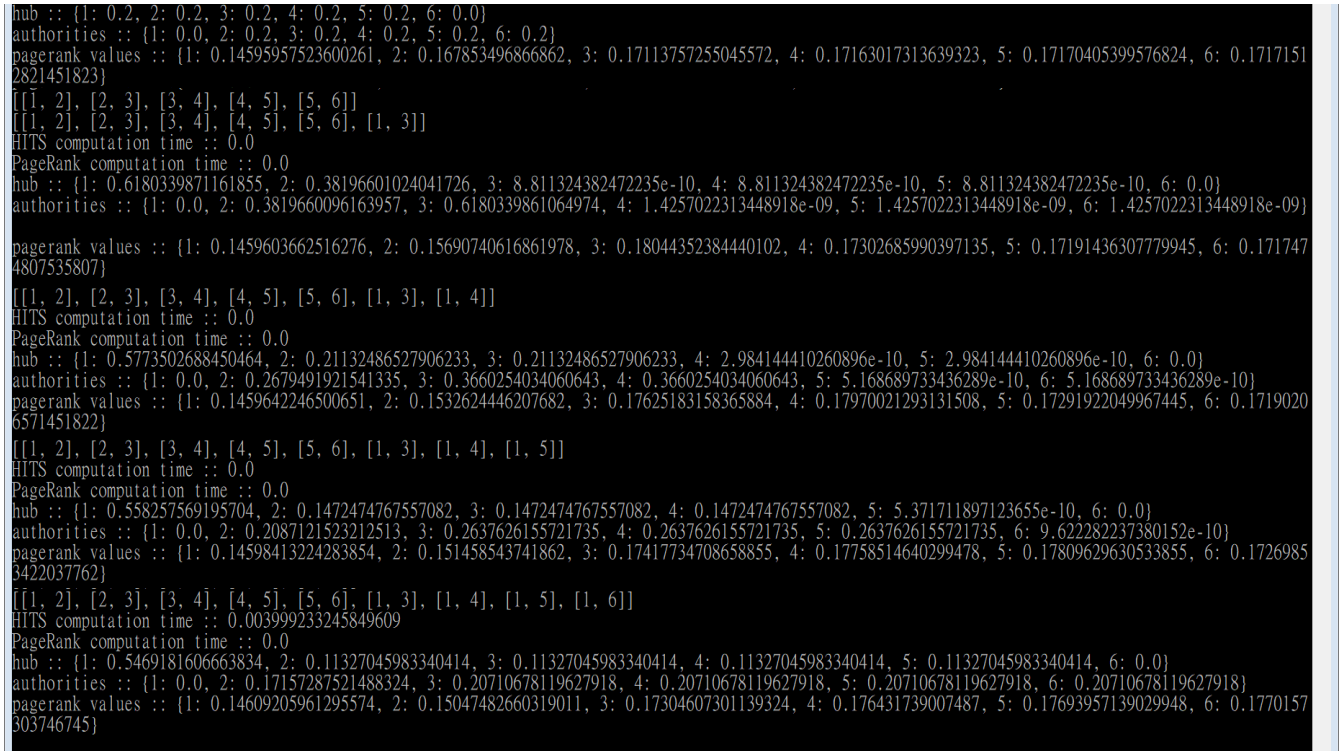
Graph 4

Graph 5 的部分則不在此呈現，有附上 csv 檔。

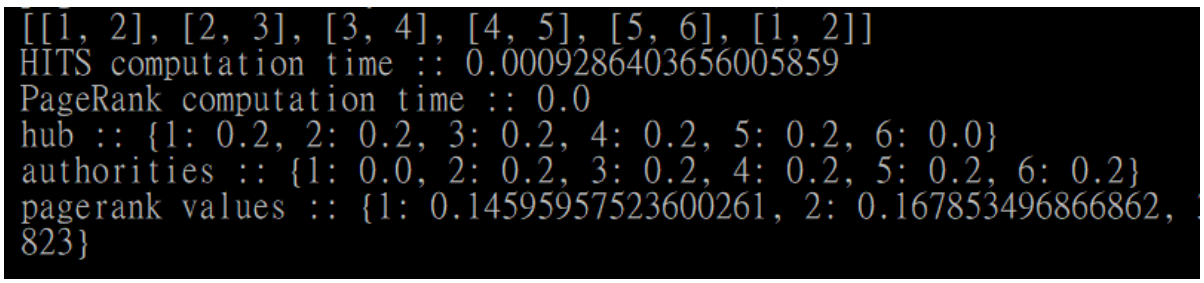
- ✧ Simrank 用於比較兩節點的相似性，核心思想為如果指向結點 a 和指向結點 b 的結點相似，那麼 a 和 b 也認為是相似的。
- ✧ Graph1 及 2 可以看出上述的概念，Graph1 而言，節點 1 指向節點 2，沒有節點指向節點 1，互相為空集合，故 simrank 得出的值為 0，Graph2 則是節點 1 指向節點 2，節點 5 指向節點 1，互相沒有交集，為空集合，故 simrank 得出的值為 0。
- ✧ Graph3 中可以稍微整理出一個例子，指向節點 1 的只有節點 2，指向節點 3 有節點 2 及節點 4，因此導致 0.666 這種值出現，Graph4 是類似的概念。

第四部份，增加 hub 值和 authority 和 pagerank 值

這一部分只有嘗試增加與節點 1 的關係，故有以下結果與結論：



- ✧ 單純增加節點 1 的指出，不一定會增加其 hub 值。
- ✧ 單純增加節點 1 地指出，小小的增加了 pagerank 值。
- ✧ 如果要增加節點的 authority，可以嘗試多設下指向該節點的關係。
- ✧ 增加 pagerank 可以嘗試增加一個單獨指向該節點但有許多其他節點只



想的節點關係。

- ✧ 嘗試增加了多重單向關係，也就是多條同樣的路，對於三種值並沒有影響。
- ✧ 嘗試增加了反向關係，也就是[1,2]相對於[2,1]，此舉突然讓節點 1 的

```
[[1, 2], [2, 3], [3, 4], [4, 5], [5, 6], [2, 1]]
HITS computation time :: 0.0019936561584472656
PageRank computation time :: 0.0
hub :: {1: 1.8626451353531692e-09, 2: 0.9999999925494194, 3: 1.862
authorities :: {1: 0.4999999962747097, 2: 1.8626451353531692e-09,
3531692e-09}
pagerank values :: {1: 0.15869062931315103, 2: 0.16976204545084636
83073}
```

hub 值大幅降低，節點 2 的 authority 大幅降低，如下圖：

PageRank 值得增加：

For graph_1.txt，增加 3 指向 1 跟 6 指向 1 後，PageRank 值增加約 0.034

```
graph_data = pd.read_csv('hw3dataset\graph_1.txt', header=None)
graph_data = graph_data.values.tolist()
graph_data.append([3, 1])
graph_data.append([6, 1])
```

```
PageRank:
[0.14595957523600261
PageRank:
[0.17914557291666666
```

For graph_2.txt，增加 3 指向 1 跟 6 指向 1 後，PageRank 值增加約 0.06

```
graph_data = pd.read_csv('hw3dataset\graph_1.txt', header=None)
graph_data = graph_data.values.tolist()
graph_data.append([3, 1])
graph_data.append([6, 1])
```

```
PageRank:
[0.14595957523600261
PageRank:
[0.20019331901041665
```

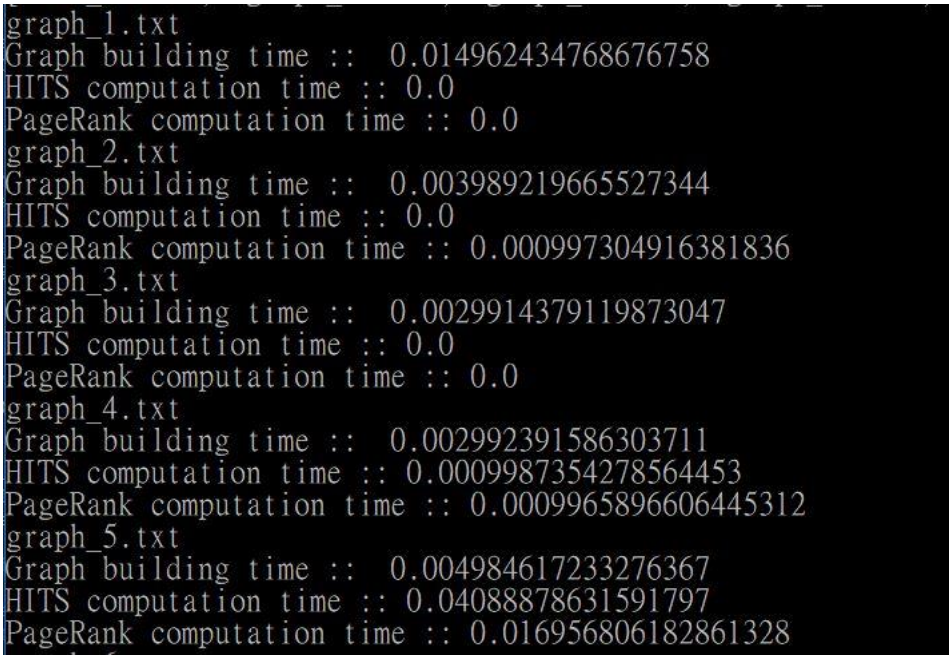
For graph_3.txt，增加 3 指向 1 跟 6 指向 1 後，PageRank 值增加約 0.08

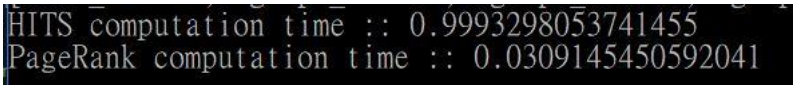
```
graph_data = pd.read_csv('hw3dataset\graph_1.txt', header=None)
graph_data = graph_data.values.tolist()
graph_data.append([3, 1])
graph_data.append([6, 1])
```

```
PageRank:
[0.14595957523600261
PageRank:
[0.22220651484374998
```

- ➢ **Computation performance analysis**

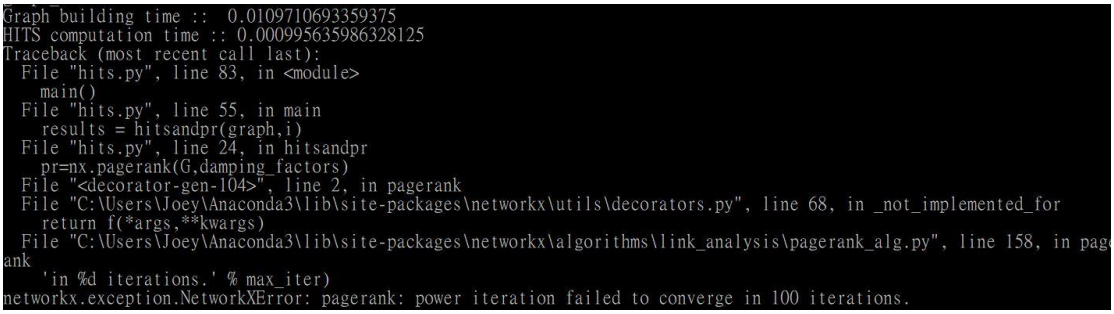
這部分做的著墨較小，只有在運算過程中稍微使用 time 函式記錄下運算時間，上面的圖也有留下紀錄，但大部分時候都因 graph 資料較小，所以被記錄下來的運算時間變成 0 了，graph5 及 6 相對較大，記錄下來的資料如下圖：





可以看到通常 PageRank 所花的時間稍微少一點。

- **Discussion**
  - ✧ 程式運算過程中發現一個小問題，就是每個 graph 所需 max_iteration 數量不同，其中 graph_6 需要最多 iteration 才可以收斂，pagerank 預設的 max_iteration 為 100，並不足以收斂，我調整至 120 才有辦法收斂



  - ✧ 在這裡先提出一個程式上可能遇到的小 bug，在做 damping_factor 的多重測試時，發現如果 damping_factor 的 datatype 是 float64，會發生以下錯誤：

```
Traceback (most recent call last):
  File "hits.py", line 59, in <module>
    main()
  File "hits.py", line 54, in main
    hubs, authorities = nx.hits(graph, max_iter=100, normalized=True)
  File "C:\Users\Joey\Anaconda3\lib\site-packages\networkx\algorithms\link_analysis\hits_alg.py", line 111, in hits
    "HITS: power iteration failed to converge in %d iterations."%(i+1))
networkx.exception.NetworkXError: HITS: power iteration failed to converge in 102 iterations.

(base) C:\Users\Joey\Desktop\資料探勘>python hits.py
['.DS_Store', 'graph_1.txt', 'graph_2.txt', 'graph_3.txt', 'graph_4.txt', 'graph_5.txt', 'graph_6.txt']
Traceback (most recent call last):
  File "hits.py", line 59, in <module>
    main()
  File "hits.py", line 54, in main
    hubs, authorities = nx.hits(graph, max_iter=110, normalized=True)
  File "C:\Users\Joey\Anaconda3\lib\site-packages\networkx\algorithms\link_analysis\hits_alg.py", line 111, in hits
    "HITS: power iteration failed to converge in %d iterations."%(i+1))
networkx.exception.NetworkXError: HITS: power iteration failed to converge in 112 iterations.
```

後將 damping_factor 調整成 np.float32 才可以正常運行。

✧ 網路上有查到一個 SimRank 的小限制，就是 SimRank 似乎無法針對無向圖運算，但無向圖可以當作 bi-directed 圖做運算，便可以突破限制。

✧ Pagerank 的 damping factor 我的測試範圍從 0.15~0.85，結果全部都記錄成 csv 檔附上，

■ Graph 1

	0.15	0.25	0.35	0.45	0.55	0.65	0.75	0.85
1	0.14596	0.132351	0.118992	0.106001	0.093524	0.08172	0.070743	0.060716
2	0.167853	0.165439	0.16064	0.153702	0.144962	0.134838	0.123799	0.112324
3	0.171138	0.173711	0.175216	0.175167	0.173253	0.169364	0.163592	0.156192
4	0.17163	0.175779	0.180319	0.184826	0.188813	0.191806	0.193437	0.19348
5	0.171704	0.176296	0.182104	0.189173	0.197371	0.206394	0.215821	0.225174
6	0.171715	0.176425	0.182729	0.19113	0.202077	0.215877	0.232608	0.252114

可以看出當 damping_factor 值調整到越高，pr 值高者會越高，低者會越低。

■ Graph 2，結果都是 0.2。

■ Graph 3

	0.15	0.25	0.35	0.45	0.55	0.65	0.75	0.85
1	0.232558	0.222222	0.212766	0.204082	0.196079	0.188679	0.181818	0.175438
2	0.267442	0.277778	0.287234	0.295918	0.303921	0.311321	0.318182	0.324562
3	0.267442	0.277778	0.287234	0.295918	0.303921	0.311321	0.318182	0.324562
4	0.232558	0.222222	0.212766	0.204082	0.196079	0.188679	0.181818	0.175438

趨勢跟 Graph 1 相同，damping_factor 值調整到越高，pr 值高者會越高，低者會越低。

- Graph 4

	0.15	0.25	0.35	0.45	0.55	0.65	0.75	0.85
1	0.169027	0.185823	0.202171	0.218158	0.233876	0.249416	0.264863	0.280288
2	0.143567	0.14424	0.145258	0.14674	0.148774	0.151427	0.154747	0.158765
3	0.13919	0.137706	0.136859	0.136548	0.136678	0.137161	0.137919	0.138882
4	0.132562	0.127113	0.122561	0.118738	0.115502	0.112735	0.110335	0.108219
5	0.161664	0.170863	0.177733	0.182507	0.185383	0.186527	0.186087	0.184199
6	0.126499	0.116434	0.107009	0.098206	0.090012	0.082424	0.075443	0.069077
7	0.127491	0.117822	0.108409	0.099104	0.089776	0.08031	0.070606	0.060571

趨勢類似於 Graph1 及 4。

✧ SimRank 的 C 測試範圍從 0.1~0.8，結果記錄成 csv 檔附上

- Graph1 及 2 都只有 1，調整 C 不影響值。
- Graph3 非 1 或 0 的值產生了變化，整理成如下的表格：

0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
0.052631	0.1111	0.176457	0.249984	0.333313	0.428543	0.538419	0.666597

C 值越大，有差異的相似節點中的**相同部分**影響較大。

- 上述的趨勢同樣出現在 graph_4 及 graph_5 中。