Link Analysis Practice

E94056178 工科系 盧柏翔

- **Implementation(詳見 main.ipynb)**

Task 1: Please implement HITS and PageRank

STEP1:讀檔並建立連結存至 list

```
for i in range(1,7):
    links = []
    with open('project3dataset/graph_' + str(i) +'.txt', 'r') as f:
        for line in f.readlines():
            link = eval('('+line.strip()+')')
            links.append(link)
```

STEP2-1:利用 python 的 networkx 套件分別計算各 Graph 的 hub、authority、pagerank 值

```
G=nx.DiGraph()
G.add_edges_from(links)
try:
    hubs, authorities = nx.hits(G,  normalized = True)
    pr = nx.pagerank(G, alpha=0.15)
except:
    print("Unexpected error:", sys.exc_info()[0])
```

STEP2-2:計算前 5 Graph 的 SimRank 值

```
for i in range(1,6):
    links = []
    with open('project3dataset/graph_' + str(i) +'.txt', 'r') as f:
        for line in f.readlines():
            link = eval('('+line.strip()+')')
            links.append(link)
    G=nx.DiGraph()
    G.add_edges_from(links)
    print(simrank(G))
    print('='*100)
```

SimRank 函式實作

```
def simrank(G, c=0.9, max_iter=100, remove_neighbors=False, remove_self=False, dump_process=False):
    sim_old = defaultdict(list)
    sim = defaultdict(list)
    for n in G.nodes():
        sim[n] = defaultdict(int)
        sim[n][n] = 1
        sim_old[n] = defaultdict(int)
        sim_old[n][n] = 0


    for iter_ctr in range(max_iter):
        if _is_converge(sim, sim_old):
            break
        sim_old = copy.deepcopy(sim)
        for i, u in enumerate(G.nodes()):
            if dump_process:
                sys.stdout.write("\r%d : % d / %d" % (iter_ctr, i, G.number_of_nodes()))
            for v in G.nodes():
                if u == v:
                    continue
                s_uv = 0.0
                for n_u in G.neighbors(u):
                    for n_v in G.neighbors(v):
                        s_uv += sim_old[n_u][n_v]
                sim[u][v] = (c * s_uv / (len(list(G.neighbors(u))) * len(list(G.neighbors(v))))) \
                    if len(list(G.neighbors(u))) * len(list(G.neighbors(v))) > 0 else 0
        if dump_process:
            print('')

    if remove_self:
        for m in G.nodes():
            G[m][m] = 0

    if remove_neighbors:
        for m in G.nodes():
            for n in G.neighbors(m):
                sim[m][n] = 0

    return sim

def _is_converge(s1, s2, eps=1e-4):
    for i in s1.keys():
        for j in s1[i].keys():
            if abs(s1[i][j] - s2[i][j]) >= eps:
                return False
    return True
```

Task 2: Find a way to increase hub, authority and PageRank of Node 1 in first 3 graphs respectively

1. 提高 Node1 hub 值(想法:增加 Node 1 到其他點的 link)

```
for i in range(1,4):
    links = []
    with open('project3dataset/graph_' + str(i) +'.txt', 'r') as f:
        for line in f.readlines():
            link = eval('('+line.strip()+')')
            links.append(link)
    G=nx.DiGraph()
    G.add_edges_from(links)
    hubs, authorities = nx.hits(G,  normalized = True)

    print('Original hub score of node 1 in Graph '+str(i)+': ',hubs[1])

    for j in range(2,7):
        links.append((1,j))
    G=nx.DiGraph()
    G.add_edges_from(links)
    hubs, authorities = nx.hits(G,  normalized = True)
    print('Revised hub score of node 1 in Graph '+str(i)+': ',hubs[1])
```

2. 提高 Node1 authority 值(想法：增加其他點到 Node 1 的 link)

```
for i in range(1,4):
    links = []
    with open('project3dataset/graph_' + str(i) +'.txt', 'r') as f:
        for line in f.readlines():
            link = eval('('+line.strip()+')')
            links.append(link)
    G=nx.DiGraph()
    G.add_edges_from(links)
    hubs, authorities = nx.hits(G,  normalized = True)

    print('Original authority score of node 1 in Graph '+str(i)+': ',authorities[1])

    for j in range(2,7):
        links.append((j,1))
    G=nx.DiGraph()
    G.add_edges_from(links)
    hubs, authorities = nx.hits(G,  normalized = True)
    print('Revised authority score of node 1 in Graph '+str(i)+': ',authorities[1])
```

3. 提高 PageRank 值(想法：盡量找 PR 值比較高且外部連結又少的點)

```
for i in range(1,4):
    links = []
    with open('project3dataset/graph_' + str(i) +'.txt', 'r') as f:
        for line in f.readlines():
            link = eval('('+line.strip()+')')
            links.append(link)
    G=nx.DiGraph()
    G.add_edges_from(links)
    pr = nx.pagerank(G, alpha=0.15)

    print('Original pagerank score of node 1 in Graph '+str(i)+': ',pr[1])

    sorted_pr = sorted(pr.items(), key=lambda d: d[1],reverse=True)
    size = int(len(sorted_pr)/2)
    sorted_pr = sorted_pr[:size]

    for k,v in sorted_pr:
        links.append((k,1))

    G=nx.DiGraph()
    G.add_edges_from(links)
    pr = nx.pagerank(G, alpha=0.15)
    print('Revised pagerank score of node 1 in Graph '+str(i)+': ',pr[1])
```

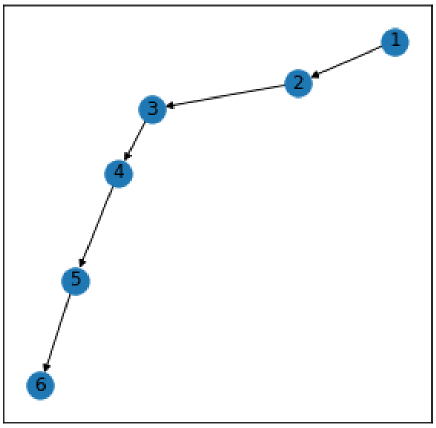
- **Result analysis and discussion**

各 graph 的 hub, authority 和 pagerank 的值

1. Graph 1

```
Graph 1:
Hub Scores:  {1: 0.2, 2: 0.2, 3: 0.2, 4: 0.2, 5: 0.2, 6: 0.0}
------------------------------------------------------------------------------------------
Authority Scores:  {1: 0.0, 2: 0.2, 3: 0.2, 4: 0.2, 5: 0.2, 6: 0.2}
------------------------------------------------------------------------------------------
PageRank value:  {1: 0.14595957523600261, 2: 0.167853496866862, 3: 0.17113757255045572, 4: 0.17163017313639323, 5: 0.17170405
399576824, 6: 0.17171512821451823}
==========================================================================================
```

分析：Graph1 圖為單向一直線，可以注意到在 Node1 因無 parent node 所以 authority 為 0，Node6 因無 chlid node 所以 hub 為 0，使用 PageRank 則可以避免此現象，因演算法賦予每個頁面最小值。

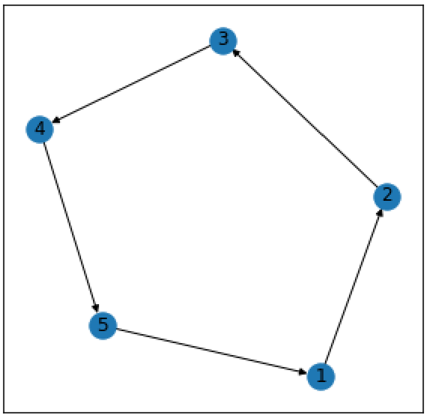


$$PR(A) = \left(\frac{PR(B)}{L(B)} + \frac{PR(C)}{L(C)} + \frac{PR(D)}{L(D)} + \cdots\right) d + \frac{1-d}{N}$$

2. Graph 2

```
Graph 2:
Hub Scores:  {1: 0.2, 2: 0.2, 3: 0.2, 4: 0.2, 5: 0.2}
------------------------------------------------------------------------------------------
Authority Scores:  {1: 0.2, 2: 0.2, 3: 0.2, 4: 0.2, 5: 0.2}
------------------------------------------------------------------------------------------
PageRank value:  {1: 0.2, 2: 0.2, 3: 0.2, 4: 0.2, 5: 0.2}
==========================================================================================
```

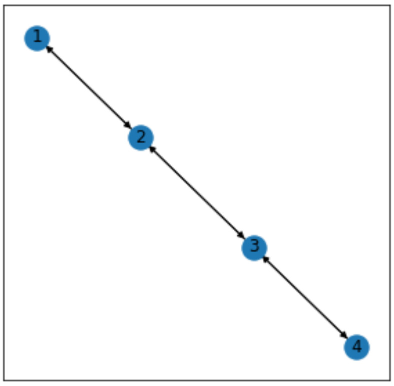
分析：Graph2 圖為單向環狀，導致 hub, authority 和 pagerank 值的結果跟初始值一樣並未改變。Hub 值為 child 的 authority 相加再歸一化，Authority 值為 parent 的 hub 相加再歸一化，pagerank 則像水流般傳值，每個點的前後狀況均相同，因此值均一樣。



3. Graph 3

```
Graph 3:
Hub Scores:  {1: 0.1909830056647784, 2: 0.3090169943352216, 3: 0.3090169943352216, 4: 0.1909830056647784}
----------------------------------------------------------------------------------------------------
Authority Scores:  {1: 0.190983005521049, 2: 0.309016994478951, 3: 0.309016994478951, 4: 0.190983005521049}
----------------------------------------------------------------------------------------------------
PageRank value:  {1: 0.23255809814453124, 2: 0.26744190185546873, 3: 0.26744190185546873, 4: 0.23255809814453124}
====================================================================================================
```

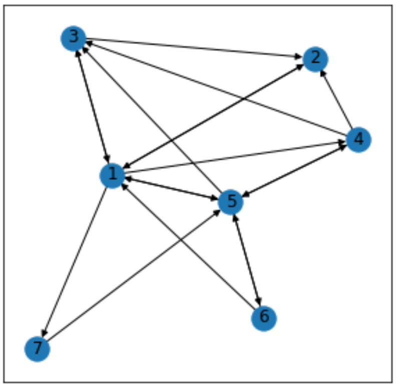
分析：Graph 3 圖為直線雙向連結，與 graph1 只有單向不同，所以各點均有值，但 node2 和 node3，indegree 和 outdegree 比 1 與 4 多，總的來說三個值都會比較高。



4. Graph 4

```
Graph 4:
Hub Scores:  {1: 0.2754531765654345, 2: 0.047762306376935765, 3: 0.10868323971440082, 4: 0.1986595564890898, 5: 0.18373459934
25369, 7: 0.06897240756733328, 6: 0.11673471394426906}
----------------------------------------------------------------------------------------------------
Authority Scores:  {1: 0.1394838930854497, 2: 0.1779120314091907, 3: 0.20082320536937326, 4: 0.1401777533243232, 5: 0.2014253
6348733986, 7: 0.08408849143863972, 6: 0.05608926188568348}
----------------------------------------------------------------------------------------------------
PageRank value:  {1: 0.16902690367606024, 2: 0.14356677737444196, 3: 0.13918989663364953, 4: 0.1325617908579799, 5: 0.1616642
6917131696, 7: 0.12649943813058034, 6: 0.12749092415597096}
====================================================================================================
```

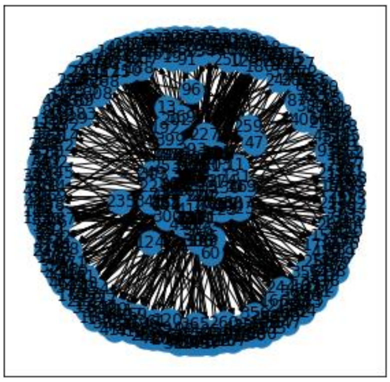
分析：從圖可看出重要的 node 像是 node5 和 node3 因為被許多 node 指向所以有較高的 authority 值，而 node 1 因為指向 node3 和 node5 較好的 node，所以 node1 的 hub 很高。而 node5 的 pagerank 高因為被好的 node 連結(像是 node1)，且連接出去的 link 少。



5. Graph 5

```
Hub Scores:  [(412, 0.02732438819668287), (325, 0.02556365229268264), (115, 0.02543914279289669), (176, 0.0279237288919764
3), (182, 0.025149235875569873), (254, 0.025697919323635597), (274, 0.028236987973692958), (293, 0.026124668497915515)]
----------------------------------------------------------------------------------------------------
Authority Scores:  [(61, 0.0958518358646769), (122, 0.09415386258073741), (134, 0.02821416470762134), (148, 0.0381169510905
47364), (185, 0.04942886096057296), (212, 0.0575704229986464), (282, 0.04971220885928309), (348, 0.0432967312721732), (104,
0.05592909137606298), (325, 0.04224824249714162)]
----------------------------------------------------------------------------------------------------
PageRank value:  [(61, 0.003861703916240950 5), (122, 0.0038212373445801816), (134, 0.002500444115252462), (148, 0.0027411084
739156267), (185, 0.002866008728224798), (212, 0.0029857139324132994), (282, 0.0029552327280543754), (348, 0.002812672353146
887), (104, 0.0031540959695848126), (96, 0.0025956098278867874), (325, 0.0028103466152241585)]
====================================================================================================
```

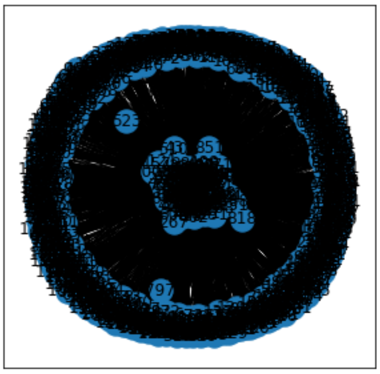
分析：此 graph 較複雜，有需多 node 但是大部分的 node 並沒有太多的 Indegree 和 Outdegree，在 weights 初始均分的情況下，大部份的值都在重要少數的 node 上，造成其他 node 的 hub, authrotiy, pagerank 都趨近為零。此處列出前幾個值較大的 node，發現 hub, authority, pagerank 的 node 都差不多。



6. Graph 6

```
Hub Scores:  [(8, 0.014477549637613262), (79, 0.015248839262942739), (386, 0.014821377482959969), (1199, 0.01506917549712052
8), (50, 0.014382924753775015), (171, 0.016151456409151662), (499, 0.015042330526635637), (755, 0.014666942605311942), (835,
0.01492474297174871), (91, 0.0152914532120512), (462, 0.014501889406065187), (185, 0.015418149804254148), (140, 0.0143127037
9600196), (169, 0.014051808642379902), (267, 0.014013032660906165), (194, 0.014445623444758296), (593, 0.01418561782782021
3), (857, 0.015518973809740825)]
-------------------------------------------------------------------------------------------
Authority Scores:  [(62, 0.03017829927899786), (78, 0.03003174252422493), (180, 0.023963146558449493), (394, 0.0293213755661
17067), (501, 0.025391313957347363), (761, 0.030404363337512435), (1123, 0.02820328654015567), (1151, 0.030404363337512435),
(819, 0.020061112957911735), (863, 0.028627191370156475), (1052, 0.024598371088986118)]
-------------------------------------------------------------------------------------------
PageRank value:  [(62, 0.0011669414455906467), (78, 0.0011475738406740473), (180, 0.0010537988113524896), (394, 0.0011369147
947026576), (501, 0.0010865479756834472), (528, 0.0010043407615147487), (761, 0.0011536698969251639), (1123, 0.00111987206126
4124), (1151, 0.0011536698969251639), (1227, 0.001012975642779283), (863, 0.0011476579696480031), (1052, 0.001234502122563268
7)]
===========================================================================================
```

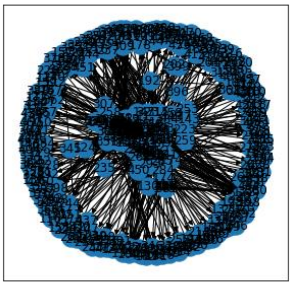
分析：此 graph 最複雜，約進行了 120 次 iteration 才達到收斂，收斂的結果跟 graph5 情況相似，大部分的 node 的 hub, authority, pagerank 都趨近零。此處列出前幾個值較大的 node，發現 hub, authority, pagerank 的 node 都差不多。



7. Graph from transaction data

```
Hub Scores:  [(34, 0.01419402051364294), (42, 0.012391057680768048), (45, 0.01209620478951702), (49, 0.014189458764042425),
(74, 0.012767200260884456), (2, 0.013543464693248478), (39, 0.012950549438727487), (20, 0.014892755864619687), (23, 0.012298
07532305994), (26, 0.014250630545078935), (96, 0.01304417145927564), (88, 0.013001222156240368), (16, 0.013356358289339542),
(92, 0.01456618775777202), (22, 0.012198297419810707), (24, 0.015218496834944431), (99, 0.012138946853505294), (37, 0.012599
556738237146), (100, 0.013790177835709143)]
-------------------------------------------------------------------------------------------
Authority Scores:  [(8, 0.022862408995193727), (36, 0.028218386619481162), (38, 0.033703953895826456), (61, 0.01681934582612
26), (63, 0.03014783098748626), (69, 0.027210640971844016), (71, 0.016749743400873904), (83, 0.01803261122067143), (11, 0.01
6391823752949626), (14, 0.016921015099345692), (17, 0.02287413377925622), (35, 0.01647636450078836), (40, 0.0187740075518832
3), (43, 0.02021421322085582), (81, 0.01928990460153051), (85, 0.0218237312844602), (87, 0.028292246569841025), (93, 0.01632
5624297943916), (3, 0.0209949823802622), (72, 0.016696085646872652), (48, 0.023082749461787833), (28, 0.023683761139743693)]
-------------------------------------------------------------------------------------------
PageRank value:  [(8, 0.011534977238857302), (36, 0.01269636667514159), (38, 0.013478505906796697), (63, 0.01311937942302217
7), (69, 0.012550641013881033), (17, 0.01147014068674206), (40, 0.01105762710951931), (43, 0.011561280264234797), (81, 0.011
229431715351094), (85, 0.011437939742637454), (87, 0.012743969918227621), (3, 0.011643305009025443), (72, 0.0110606276183668
07), (48, 0.01151432024791988), (28, 0.011921908741664263)]
===========================================================================================
```

分析：由 transaction data 產生的 graph，大部分的 node 未與其他 node 連接，因此跟前兩個 graph 相似，node 多導致值小且集中在少數 node 上。



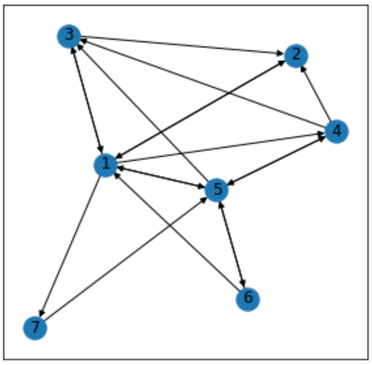
SimRank(Graph5 太大不列，詳見 main.ipynb)

```
Graph1:
defaultdict(<class 'list'>, {1: defaultdict(<class 'int'>, {1: 1, 2: 0.0, 3: 0.0, 4: 0.0, 5: 0.0, 6: 0}), 2: defaultdict(<cla
ss 'int'>, {2: 1, 1: 0.0, 3: 0.0, 4: 0.0, 5: 0.0, 6: 0}), 3: defaultdict(<class 'int'>, {3: 1, 1: 0.0, 2: 0.0, 4: 0.0, 5: 0.
0, 6: 0}), 4: defaultdict(<class 'int'>, {4: 1, 1: 0.0, 2: 0.0, 3: 0.0, 5: 0.0, 6: 0}), 5: defaultdict(<class 'int'>, {5: 1,
1: 0.0, 2: 0.0, 3: 0.0, 4: 0.0, 6: 0}), 6: defaultdict(<class 'int'>, {6: 1, 1: 0, 2: 0, 3: 0, 4: 0, 5: 0})})
==========================================================================================
Graph2:
defaultdict(<class 'list'>, {1: defaultdict(<class 'int'>, {1: 1, 2: 0.0, 3: 0.0, 4: 0.0, 5: 0.0}), 2: defaultdict(<class 'in
t'>, {2: 1, 1: 0.0, 3: 0.0, 4: 0.0, 5: 0.0}), 3: defaultdict(<class 'int'>, {3: 1, 1: 0.0, 2: 0.0, 4: 0.0, 5: 0.0}), 4: defau
ltdict(<class 'int'>, {4: 1, 1: 0.0, 2: 0.0, 3: 0.0, 5: 0.0}), 5: defaultdict(<class 'int'>, {5: 1, 1: 0.0, 2: 0.0, 3: 0.0,
4: 0.0})})
==========================================================================================
Graph3:
defaultdict(<class 'list'>, {1: defaultdict(<class 'int'>, {1: 1, 2: 0.0, 3: 0.8181254024807277, 4: 0.0}), 2: defaultdict(<cl
ass 'int'>, {2: 1, 1: 0.0, 3: 0.0, 4: 0.8181254024807277}), 3: defaultdict(<class 'int'>, {3: 1, 1: 0.8181254024807277, 2: 0.
0, 4: 0.0}), 4: defaultdict(<class 'int'>, {4: 1, 1: 0.0, 2: 0.8181254024807277, 3: 0.0})})
==========================================================================================
```

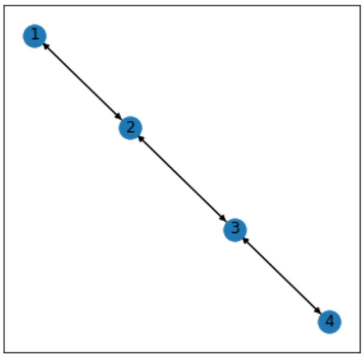

```
Graph3:
defaultdict(<class 'list'>, {1: defaultdict(<class 'int'>, {1: 1, 2: 0.0, 3: 0.8181254024807277, 4: 0.0}), 2: defaultdict(<cl
ass 'int'>, {2: 1, 1: 0.0, 3: 0.0, 4: 0.8181254024807277}), 3: defaultdict(<class 'int'>, {3: 1, 1: 0.8181254024807277, 2: 0.
0, 4: 0.0}), 4: defaultdict(<class 'int'>, {4: 1, 1: 0.0, 2: 0.8181254024807277, 3: 0.0})})
==========================================================================================
```


```
Graph4:
defaultdict(<class 'list'>, {1: defaultdict(<class 'int'>, {1: 1, 2: 0.49914978795239817, 3: 0.5432562826894801, 4: 0.5895017
588318365, 5: 0.5626058248027922, 7: 0.5788949956670428, 6: 0.5390223918097206}), 2: defaultdict(<class 'int'>, {2: 1, 1: 0.4
9914978795239817, 3: 0.674580207831517, 4: 0.4814362341216265, 5: 0.6011029722009221, 7: 0.5062838874087376, 6: 0.70314194370
43688}), 3: defaultdict(<class 'int'>, {3: 1, 1: 0.5432562826894802, 2: 0.6745802078315171, 4: 0.5820541017139937, 5: 0.56583
21394920336, 7: 0.5236105011583361, 6: 0.5990953544949266}), 4: defaultdict(<class 'int'>, {4: 1, 1: 0.5895017588318365, 2:
0.4814362341216265, 3: 0.5820541017139936, 5: 0.5482348511142967, 7: 0.6500414699808235, 6: 0.565738852051225}), 5: defaultdi
ct(<class 'int'>, {5: 1, 1: 0.5626058248027922, 2: 0.6011029722009221, 3: 0.5658321394920335, 4: 0.5482348511142968, 7: 0.501
1970971241524, 6: 0.5511500346625372}), 7: defaultdict(<class 'int'>, {7: 1, 1: 0.5788949956670428, 2: 0.5062838874087378, 3:
0.5236105011583361, 4: 0.6500414699808235, 5: 0.5011970971241524, 6: 0.7031419437043689}), 6: defaultdict(<class 'int'>, {6:
1, 1: 0.5390223918097206, 2: 0.7031419437043689, 3: 0.5990953544949266, 4: 0.565738852051225, 5: 0.5511500346625373, 7: 0.703
1419437043688})})
==========================================================================================
```

Simrank 是評估任意兩節點相似度的演算法，主要思想為兩個物體與相似物體有關聯，那麼這兩個物體是相似的。通過結點之間的結構化的相同點來計算結點之間的相似度。



以左圖 Graph4 為例，因 node6 與 node7 均有指向 node5 所以相似度較高為 0.703。



以左圖 Graph3 為例，node1 和 node3 均指向與被指向 node2 相似度高為 0.818；同樣 node2 和 node4 也是相同情形相似度為 0.818。

Node1 增加 hub, authority 和 pagerank

提升 hub 主要思想為盡量將自身網站連向具有高 authority 的網站，因此實作主要將 node1 連接出去；提升 authority 主要思想為盡量將具有高 hub 的網站連向 node1，因此實作主要將其他 node 連接進來；提升 pagerank 主要思想為盡量讓 pagerank 值高的網站連向 node1，因此實作主要將前幾個高 pagerank 的 node 連向 node1。

Computation performance analysis

Graph 越複雜耗時越久

HITS	Graph1	Graph2	Graph3	Graph4	Graph5	Graph6	Graph7
Time(sec)	0	0	0	0.001	0.053856	1.368556	1.108543
iteration	2	2	11	20	19	120	73

PageRank	Graph1	Graph2	Graph3	Graph4	Graph5	Graph6	Graph7
Time(sec)	0	0.000997	0	0.001	0.007978	0.024933	0.039893

SimRank	Graph1	Graph2	Graph3	Graph4	Graph5
Time(sec)	0	0	0.000997	0.00598	3.933138

Discussion

More limitations about link analysis algorithms

Answer:

Hits 的限制—依照邏輯，你不需要是位優秀的作家，才能評論其他作家，你只要是一位備受肯定的評論家就行了。HITS 算法還存在易被作弊者操縱結果，結構不穩定等問題。

Pagerank 的限制—沒有區分網站內的相互連結、沒有過濾廣告連結和功能連結、對新網頁不利。

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

Answer: 並不一定，若是存在許多垃圾網頁指向某網域，可能會造成某網頁是重要的假象，而且常常會受一些廣告連結干擾。

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

Answer: 實際各個網頁是動態的，不容易收斂變穩定，且太多的廣告連結會干擾分析效果。

pagerank 在實際應用中，Web 連接圖中常常存在一些 indegree 或 outdegree 為 0 的 node，這時會出現兩種異常：等級變化（RankLeak）和等級下沉（RankSink）

Any new idea about the link analysis algorithm?

Answer: 先人工決定較好或較信任的網站，信任度最高，之後向外連接逐漸下降，主要思想是好的網站很少連到壞的網站，但相反不成立，希望藉此找出更多好的網站。

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

Answer: SimRank 公式如下，參數 C 像是阻尼係數，可以這樣理解：假如 I（a）= I（b）= {A}，按照（1）式計算出 sim（a，b）= C * sim（A，A）= C，所以 C∈（0， 1），C 的大小影響 Simrank value 高低。

$$s(a,b) = \frac{C}{|I(a)||I(b)|} \sum_{i=1}^{|I(a)|} \sum_{j=1}^{|I(b)|} s(I_i(a), I_j(b))$$