Data Mining

Project 3 - Link Analysis Practice

- 沈育同
- P76061386

Environment

- Ubuntu 18.04.1 LTS (GNU/Linux 4.15.0-34-generic x86_64)

Prerequisite

- Python 3.6.4

Install Dependency

```
$ pip install -r requirements.txt
```

Usage

```
$ python main.py [-h] [-f FILE]
```

optional Options	Description
-h, --help	show this help message and exit
-f FILE	Place the input into <file>

Files Structure

```
.
+-- hw3dataset
|   +-- graph_1.txt
|   +-- graph_2.txt
|   +-- graph_3.txt
|   +-- graph_4.txt
|   +-- graph_5.txt
|   +-- graph_6.txt
|   +-- graph_7.txt
|   +-- graph_8.txt
|   +-- graph_ex.txt
|   +-- IBM_data.txt
|   +-- Read_me.txt
+-- result
|   +-- auth_1.png
|   +-- auth_2.png
|   +-- auth_3.png
|   +-- auth_4.png
|   +-- auth_5.png
|   +-- auth_6.png
|   +-- hub_1.png
|   +-- hub_2.png
|   +-- hub_3.png
|   +-- hub_4.png
|   +-- hub_5.png
|   +-- hub_6.png
|   +-- pr_1.png
|   +-- pr_2.png
|   +-- pr_3.png
|   +-- pr_4.png
|   +-- pr_5.png
|   +-- pr_6.png
|   +-- simrank_5_c08.txt
|   +-- simrank_5_c10.txt
+-- requirements.txt
+-- trans2graph.py
+-- ggen.py
+-- main.py
```

- `hw3dataset/`：本次 project 實驗 graph 檔案。
- `result/`：實驗結果及圖表。
- `requirements.txt`：python套件需求。
- `trans2graph.py`：將 IBM transaction data 轉換成 graph 之輔助程式。
- `ggen.py`：graph 生成程式。
- `main.py`：主程式。

Implement

- HITS
  - 使用 list 結構儲存每個節點的 Authority 及 Hub 值，並初始化為 1 / n ^ 0.5
  - 進入以下迴圈
    - 對於每個節點，
      - 將其 Authority 更新為其全部父節點之 Hub 值總和。
      - 將其 Hub 更新為其全部子節點之 Authority 值總和。
    - 將全部 Authority 除以全部 Authority 平方和開根號 (歐基理德距離)
    - 將全部 Hub 除以全部 Hub 平方和開根號 (歐基理德距離)
    - 如果新舊 Authority 之差值平方和，加上新舊 Hub 之差值平方和小於 epsilon (default: 1e-10)，則跳出迴圈。
  - 輸出 Authority 及 Hub list。
- PageRank
  - 使用 list 結構儲存每個節點的 PageRank 值，並初始化為 1 / n ^ 0.5

    - 進入以下迴圈
        - 對於每個節點，
            - 將其 PageRank 更新為 d 除以總節點數，加上 (1-d) 乘以其全部父節點 PageRank 值除以父節點對外分支數之總和。 (default: d=0.1)
        - 將全部 PageRank 除以全部 PageRank 平方和開根號 (歐基理德距離)
        - 如果新舊 PageRank 之差值平方和小於 epsilon (default: 1e-10)，則跳出迴圈。
    - 輸出 PageRank list。

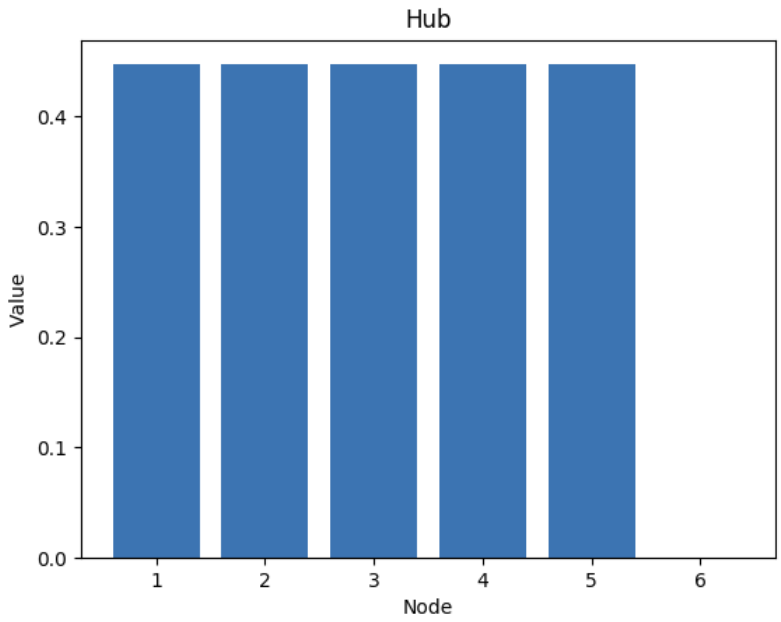
- SimRank
    - 建立一個 dictionary 來儲存節點與節點之間的倆倆 SimRank 值，並初始化自己對自己的 SimRank 為 1 ，其餘的 SimRank 為 0 。
    - 進入以下迴圈
        - 對於全部節點中的倆倆配對 a, b 節點，其中 a 不等於 b ，
            - 將 SimRank(a, b) 及 SimRank(b, a) 更新為 C 除以 a 之父節點數，除以 b 之父節點數，乘以全部 a 之父節點配對上全部 b 之父節點之 SimRank 總和。 (default: C=1.0)
        - 如果更新前後 SimRank 之差值平方和小於 epsilon (default: 1e-10)，則跳出迴圈。
    - 輸出 SimRank dictionary。

Result

- HITS
    - Graph1

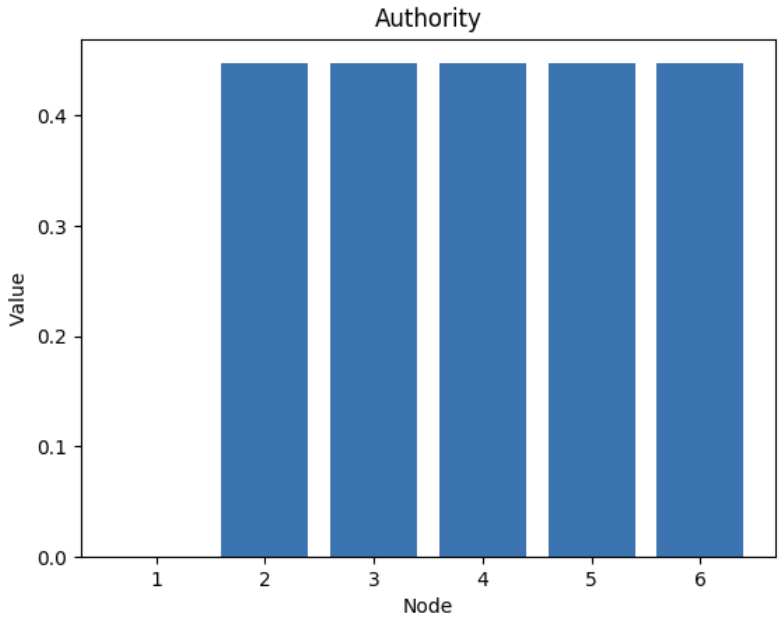
      ```
      1 -> 2 -> 3 -> 4 -> 5 -> 6
      ```

        - Hub



	Node	Value
Top-1	1	0.44721
Top-2	2	0.44721
Top-3	3	0.44721
Top-4	4	0.44721
Top-5	5	0.44721

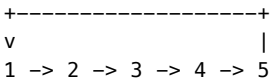
- Authority



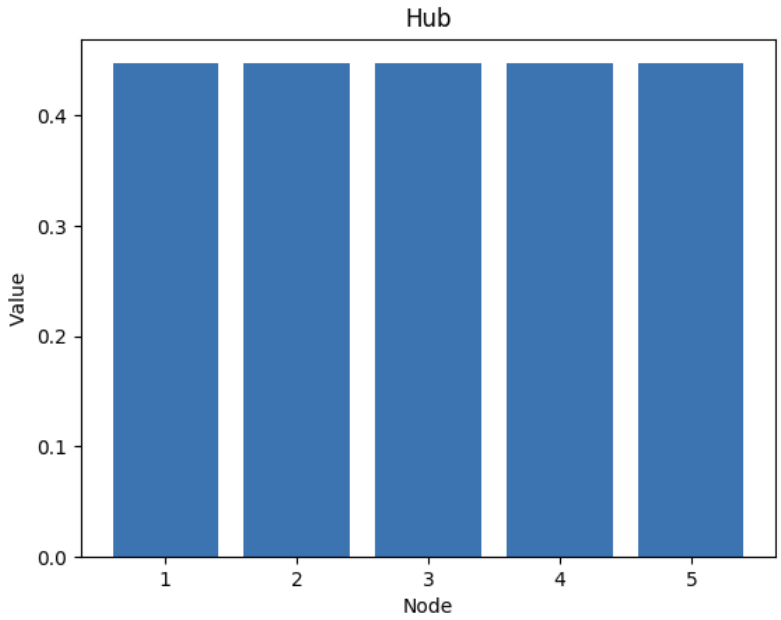
	Node	Value
Top-1	2	0.44721
Top-2	3	0.44721
Top-3	4	0.44721
Top-4	5	0.44721
Top-5	6	0.44721

- Discussion
  - 由節點連線關係可以看的出來 Node1 並沒有被其他節點指向(推薦)，因此其 Authority 值為 0 ，其他節點都可以接收其父節點之 Hub 值為其 Authority 值。
  - 由節點連線關係可以看的出來 Node6 並沒有指向(推薦)其他節點，因此其 Hub 值為 0 ，其他節點都可以接收其子節點之 Authority 值為其 Hub 值。
  - 因此說明了如果對外指向的節點數愈多 Hub 就會有機會愈高，如果被愈多節點指向則 Authority 將會有機會愈高。

- Graph2

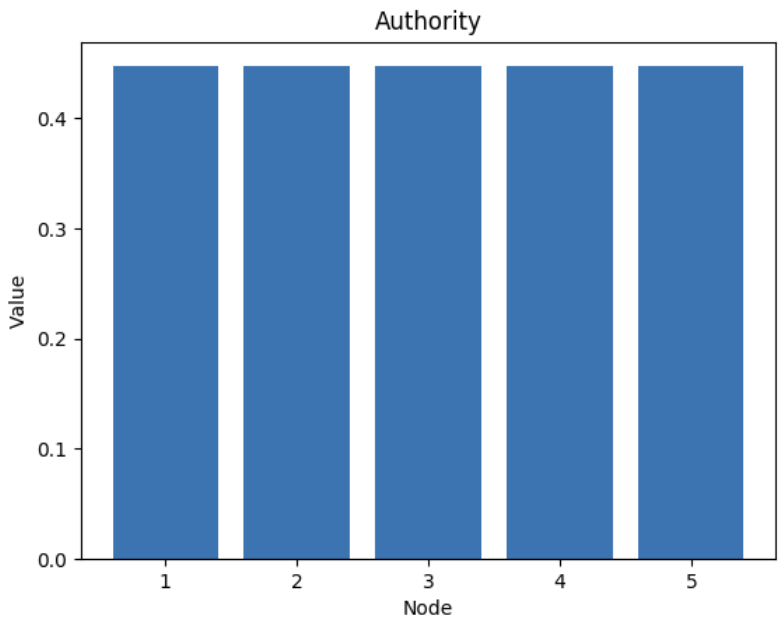


- Hub



	Node	Value
Top-1	1	0.44721
Top-2	2	0.44721
Top-3	3	0.44721
Top-4	4	0.44721
Top-5	5	0.44721

- Authority



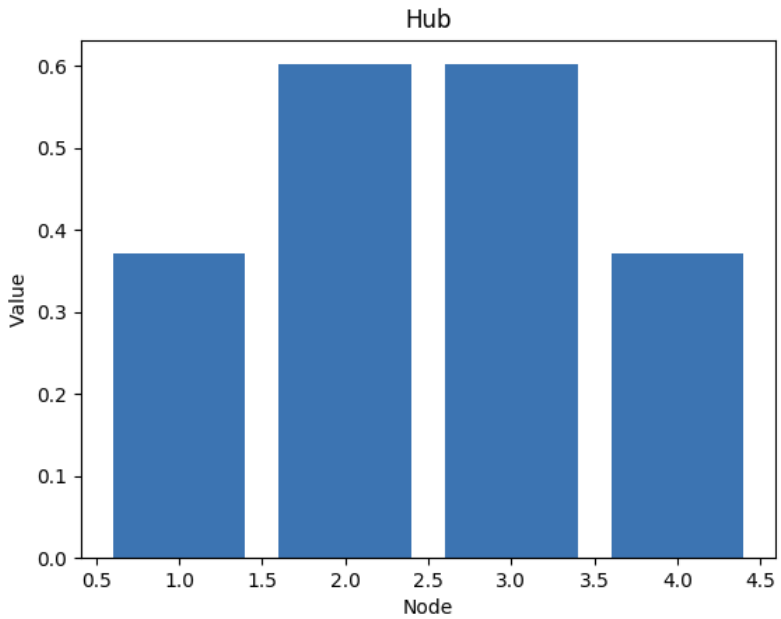
	Node	Value
Top-1	1	0.44721
Top-2	2	0.44721
Top-3	3	0.44721
Top-4	4	0.44721
Top-5	5	0.44721

- Discussion
  - 因為這個 Graph 是一個環狀，每個節點都是被前一個節點指向，同時也指向下一個節點，沒有任何差異，因此不論在 Hub 及 Authority 皆為一致。
  - 如果將此環狀架構打破一個 edge 那麼就會跟 graph1 一樣，最末端的節點 Hub 值為 0 ，最前端節點 Authority 為 0。

- Graph3

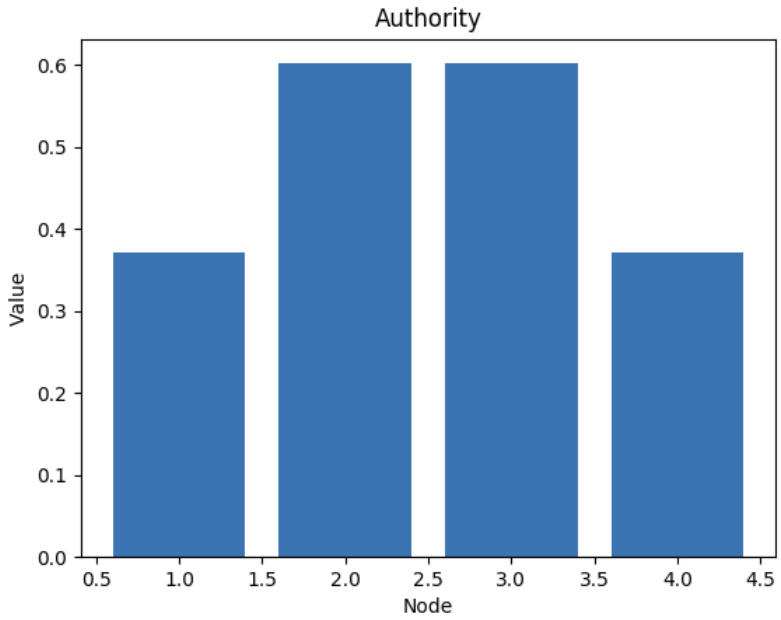
```
1 <-> 2 <-> 3 <-> 4
```

- Hub



	Node	Value
Top-1	2	0.60150
Top-2	3	0.60150
Top-3	1	0.37175
Top-4	4	0.37175

- Authority



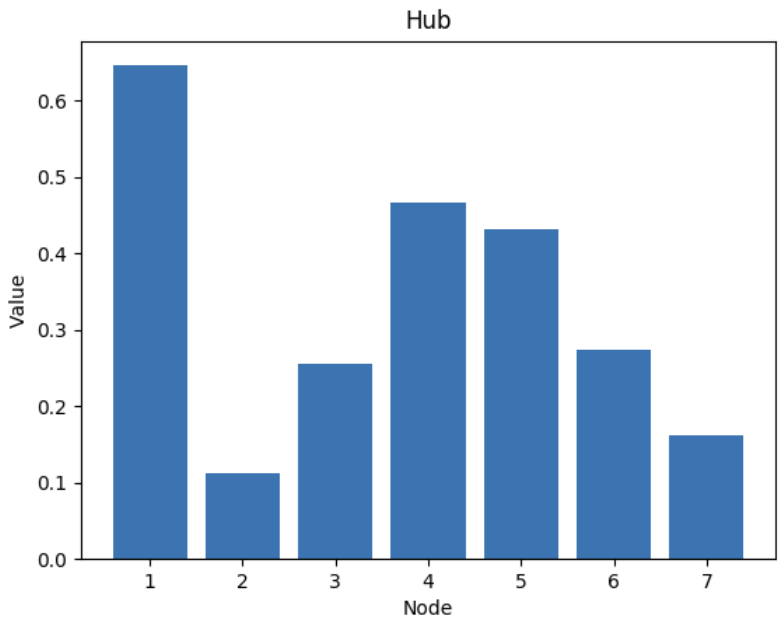
	Node	Value
Top-1	2	0.60150
Top-2	3	0.60150
Top-3	1	0.37175
Top-4	4	0.37175

- Discussion
  - 觀察 graph3 ，若將 Node2 與 Node3 中間之 edge 當分界分為兩個子圖，會發現 Node2 這側的子圖與 Node3 這側的子圖是一模一樣，因此可以預期其 Hub 與 Authority 會有對稱的情形發生。
  - 以 Node1 來說其對外只指向 Node2 ，因此其 Hub 只能承接 Node2 的 Authority ，但是 Node2 同時指向 Node1 及 Node3 ，因此其 Hub 承接了 Node1 及 Node3 的 Authority ，因此 Node2 之 Hub 高於 Node1。
  - 另外，以 Node1 來說其只被 Node2 指向，因此其 Authority 只能承接 Node2 的 Hub ，但是 Node2 同時被 Node1 及 Node3 指向，因此其 Authority 承接了 Node1 及 Node3 的 Hub ，因此 Node2 之 Authority 高於 Node1。
  - 整體架構因為也是對稱的狀況，Node1 對應上 Node4 及 Node2 對應上 Node3 ，因此 Node1 及 Node4 之 Hub 與 Authority 互相一致， Node2 及 Node3 之 Hub 與 Authority 互相一致。

- Graph4

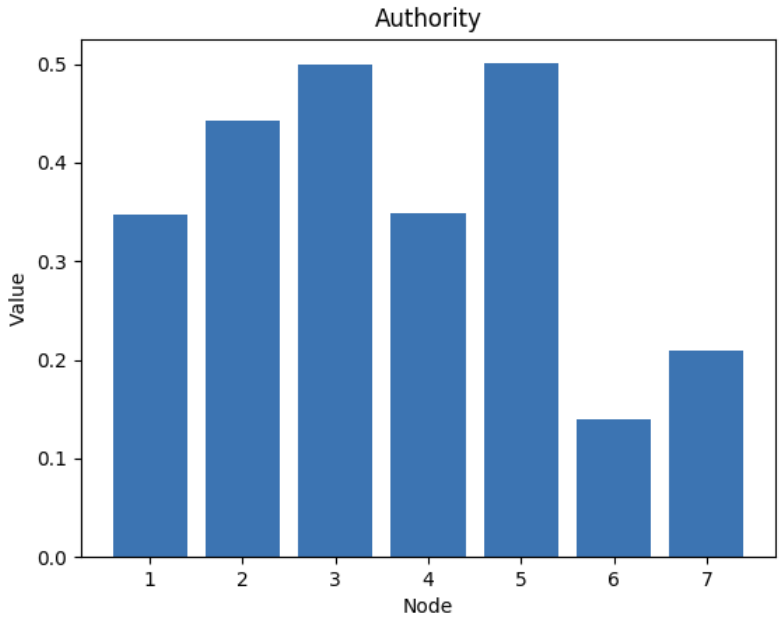
```
Node 1
In  Node: [2, 3, 5, 6]
Out Node: [2, 3, 4, 5, 7]
Node 2
In  Node: [1, 3, 4]
Out Node: [1]
Node 3
In  Node: [1, 4, 5]
Out Node: [1, 2]
Node 4
In  Node: [1, 5]
Out Node: [2, 3, 5]
Node 5
In  Node: [1, 4, 6, 7]
Out Node: [1, 3, 4, 6]
Node 7
In  Node: [1]
Out Node: [5]
Node 6
In  Node: [5]
Out Node: [1, 5]
```

- Hub



	Node	Value
Top-1	1	0.64642
Top-2	4	0.46621
Top-3	5	0.43119
Top-4	6	0.27395
Top-5	3	0.25506

- Authority

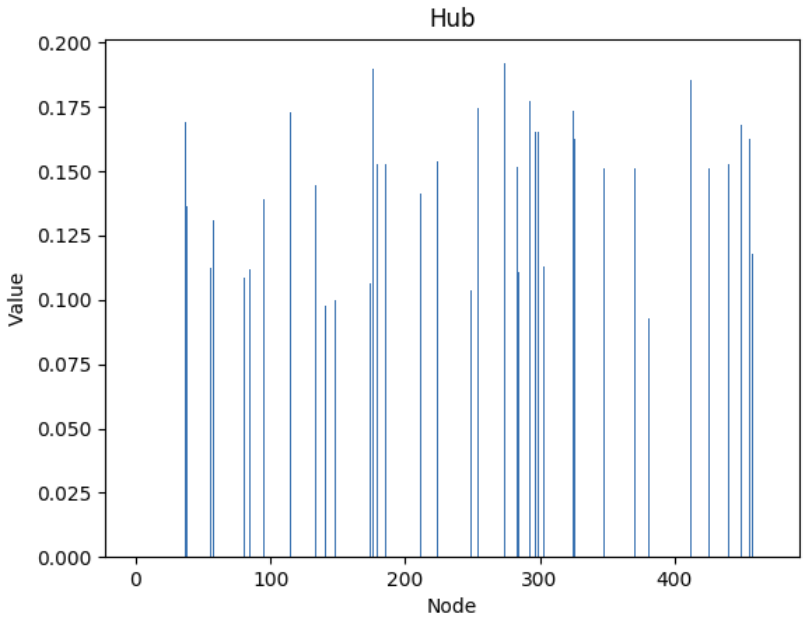


	Node	Value
Top-1	5	0.50063
Top-2	3	0.49914
Top-3	2	0.44219
Top-4	4	0.34841
Top-5	1	0.34669

- Discussion
  - 以 hub 值來看，由於 Node1 推薦了 Node2, 3, 4, 5, 7 這些高 authority 的 Node 因此其 hub 值相較於其他節點是最高的。反之 Node2 只推薦了 Node1 而 Node1 卻非很高的 authority 因此 Node2 的 hub 值相較於其他節點是最低的。
  - 以 authority 來說， Node3, 5 分別被 Node1, 4, 5 及 Node1, 4, 6, 7 推薦，其中 Node1, 4 分別為 hub 排名前兩名高的， Node5 之 hub 大約與 Node6, 7 之 hub 相加一致，因此 Node3, 5 相較於其他節點 authority 是高的。反之，Node6 只被 Node5 推薦，而 Node5 之 hub 值又並非特別高，因此 Node6 之 authority 相較其他節點為最低的。

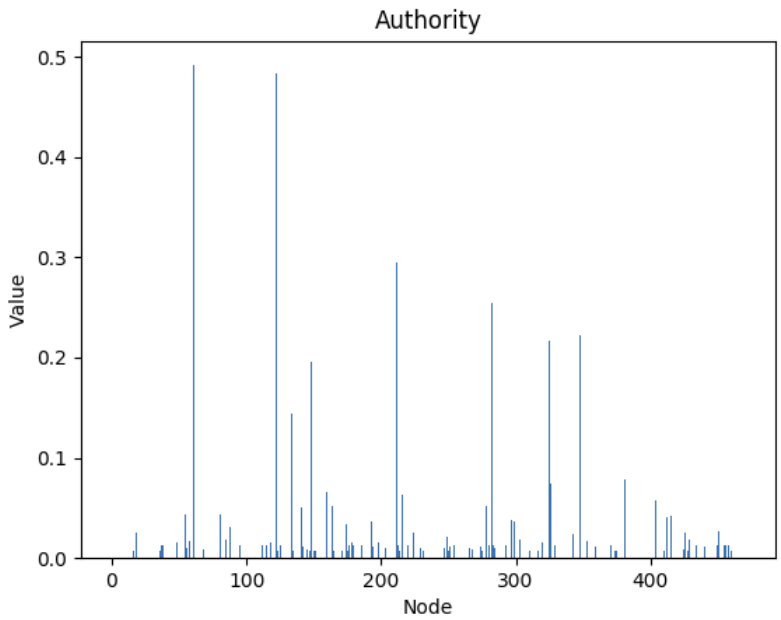
- Graph5

- Hub



	Node	Value
Top-1	274	0.19194
Top-2	176	0.18981
Top-3	412	0.18574
Top-4	293	0.17759
Top-5	254	0.17468
Top-6	325	0.17377
Top-7	115	0.17293
Top-8	182	0.17095
Top-9	37	0.16909
Top-10	450	0.16789

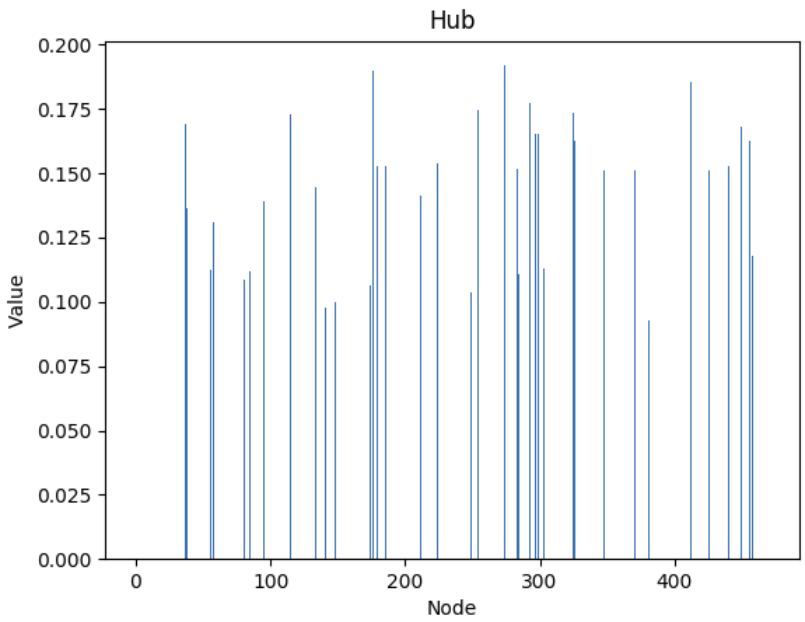
- Authority



	Node	Value
Top-1	61	0.49135
Top-2	122	0.48265
Top-3	212	0.29511
Top-4	104	0.28670
Top-5	282	0.25483
Top-6	185	0.25338
Top-7	348	0.22195
Top-8	325	0.21657
Top-9	148	0.19539
Top-10	134	0.14463

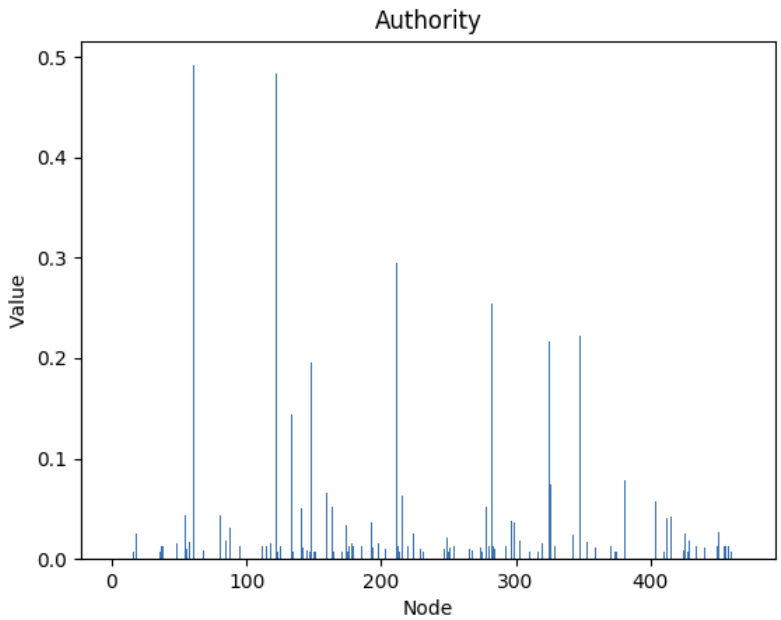
- Discussion
  - 以 hub 值來看， Node274 推薦了 Node61, 104, 122, 130, 148, 160, 185, 212, 315, 325, 348, 381 幾乎把排名前10名 authority 的 Node 都推薦到了，因此 Node274 拿到了最高的 hub 值，至於那些完全沒有推薦其他節點的節點其 hub 值都為 0 。
  - 以 authority 來說， Node61 被 Node37, 38, 56, 58, 81, 85, 95, 103, 115, 130, 134, 141, 148, 174, 176, 179, 182, 184, 185, 186, 212, 224, 236, 249, 254, 261, 274, 283, 284, 285, 293, 297, 299, 303, 315, 325, 326, 348, 371, 381, 412, 416, 426, 440, 443, 450, 456, 458 推薦，排名前10名 hub 的 Node 都有推薦到 Node61 ，因此相較了其他節點 Node61 拿到了最高的 authority ，至於那些完全沒有被其他節點推薦的節點其 authority 值都為 0 。

- Graph6
  - Hub



	Node	Value
Top-1	171	0.15626
Top-2	857	0.15014
Top-3	185	0.14917
Top-4	91	0.14794
Top-5	79	0.14753
Top-6	1199	0.14579
Top-7	499	0.14553
Top-8	835	0.14439
Top-9	386	0.14339
Top-10	755	0.14190

- Authority



	Node	Value
Top-1	761	0.27506
Top-2	1151	0.27506
Top-3	62	0.27302
Top-4	78	0.27169
Top-5	394	0.26527
Top-6	863	0.25899
Top-7	1123	0.25515
Top-8	501	0.22971
Top-9	1052	0.22255
Top-10	180	0.21679

- Discussion

- 以 hub 值來看， Node171 推薦了 Node62, 78, 91, 180, 205, 211, 218, 225, 348, 350, 357, 364, 384, 386, 394, 440, 483, 501, 506, 521, 528, 581, 592, 622, 640, 761, 793, 815, 819, 829, 846, 863, 867, 929, 931, 939, 1052, 1054, 1071, 1098, 1113, 1123, 1147, 1151, 1184, 1199, 1227 完全把排名前10名 authority 的 Node 都推薦到了，因此 Node171 拿到了最高的 hub 值，至於那些完全沒有推薦其他節點的節點其 hub 值都為 0 。
- 以 authority 來說， Node761 被 Node7, 8, 9, 25, 50, 59, 62, 78, 79, 81, 91, 134, 140, 169, 171, 180, 185, 187, 194, 202, 250, 266, 267, 300, 309, 312, 336, 382, 386, 397, 441, 449, 459, 462, 499, 501, 506, 540, 552, 580, 581, 593, 606, 641, 659, 662, 739, 755, 803, 815, 835, 857, 863, 896, 905, 914, 929, 930, 944, 996, 1019, 1071, 1113, 1131, 1150, 1157, 1199, 1227 推薦，排名前10名 hub 的 Node 都有推薦到 Node761 ，因此相較了其他節點， Node761 拿到了最高的 authority ，至於那些完全沒有被其他節點推薦的節點其 authority 值都為 0 。

- Graph7

> 本資料的構成是由 project1 之 IBM dataset 構成，將每個 transaction 的各個 item 以單向連接到下一個 item，由於總節點數為 7030 ，總邊數為 8487 無法使用圖形呈現，詳細 graph 詳見 `hw3dataset/graph7.txt` 。

  - Hub

	Node	**Value**
Top-1	63977	0.63891
Top-2	53991	0.31904
Top-3	55424	0.27017
Top-4	80473	0.21575
Top-5	81578	0.21301
Top-6	50609	0.20891
Top-7	69962	0.17264
Top-8	58531	0.16614
Top-9	64308	0.16614
Top-10	73402	0.16018
Top-11	73044	0.14514
Top-12	62898	0.13878
Top-13	51873	0.13785
Top-14	69846	0.12022
Top-15	60307	0.11854
Top-16	65849	0.10810
Top-17	57341	0.10741
Top-18	61398	0.10741
Top-19	62599	0.10741
Top-20	63765	0.10741

  - Authority

	Node	Value
Top-1	67992	0.54218
Top-2	83398	0.47363
Top-3	63977	0.35051
Top-4	69962	0.26575
Top-5	67935	0.21607
Top-6	71779	0.19578
Top-7	73044	0.19578
Top-8	95785	0.19578
Top-9	89998	0.16434
Top-10	55424	0.14845
Top-11	78129	0.09236
Top-12	93157	0.08974
Top-13	87458	0.08418
Top-14	66198	0.08279
Top-15	71149	0.07556
Top-16	84006	0.07204
Top-17	98167	0.06611
Top-18	82155	0.06527
Top-19	55189	0.06402
Top-20	65849	0.05389

- Discussion
  - 由 node 及 edge 比例來看，可以得知平均一個節點大約一個至兩個 in-node 及 out-node。
  - 以 hub 值來看， Node63977 推薦了 Node67935, 83398, 73044, 95785, 67992, 71779, 69962 幾乎把排名前10名 authority 的 Node 都推薦到了，因此 Node63977 拿到了最高的 hub 值，至於那些完全沒有推薦其他節點的節點其 hub 值都為 0 。
  - 以 authority 來說， Node67992 被 Node53991, 58531, 64308, 63977, 50609, 55424 推薦，排名前6名 hub 的 Node 除了第5名的 Node81578 以外，都推薦到 Node67992 ，因此相較了其他節點， Node67992 拿到了最高的 authority ，至於那些完全沒有被其他節點推薦的節點其 authority 值都為 0 。

- Graph8

  本資料的構成是由 project1 之 IBM dataset 構成，將每個 transaction 的各個 item 以雙向連接到下一個 item，由於總節點數為 7030 ，總邊數為 16974 無法使用圖形呈現，詳細 graph 詳見 `hw3dataset/graph8.txt` 。

  - Hub

	Node	Value
Top-1	63977	0.47205
Top-2	67992	0.29230
Top-3	83398	0.27000
Top-4	53991	0.23938
Top-5	37407	0.20800
Top-6	51873	0.19640
Top-7	69962	0.18902
Top-8	62898	0.17381
Top-9	55424	0.16724
Top-10	73044	0.14652
Top-11	60307	0.14616
Top-12	47626	0.14266
Top-13	80473	0.13548
Top-14	57341	0.13428
Top-15	36978	0.12043
Top-16	39721	0.11080
Top-17	71779	0.11044
Top-18	67935	0.10598
Top-19	81578	0.10550
Top-20	61398	0.09834

- Authority

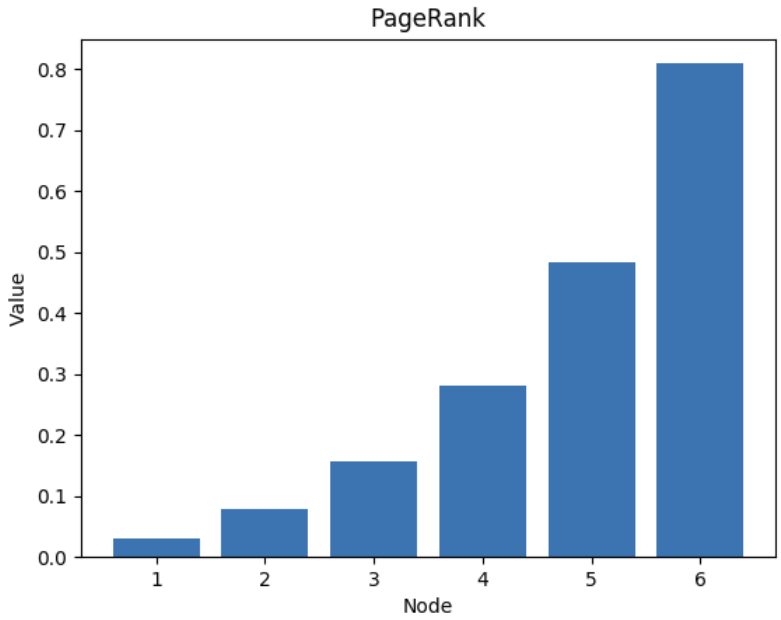
	Node	Value
Top-1	63977	0.47205
Top-2	67992	0.29230
Top-3	83398	0.27000
Top-4	53991	0.23938
Top-5	37407	0.20800
Top-6	51873	0.19640
Top-7	69962	0.18902
Top-8	62898	0.17381
Top-9	55424	0.16724
Top-10	73044	0.14652
Top-11	60307	0.14616
Top-12	47626	0.14266
Top-13	80473	0.13548
Top-14	57341	0.13428
Top-15	36978	0.12043
Top-16	39721	0.11080
Top-17	71779	0.11044
Top-18	67935	0.10598
Top-19	81578	0.10550
Top-20	61398	0.09834

        - Discussion
            - 由於 graph8 架構之 edge 為雙向，有 edge a 連到 b 就會有 edge b 連到 a，因此 hub 與 authority 之更新都是一致的。
            - 由 node 及 edge 比例來看，可以得知平均一個節點大約兩個至三個 in-node 及 out-node。
            - Node63977 推薦了 Node61398, 67935, 63765, 83398, 57341, 73044, 60307, 62599, 95785, 53991, 67992, 51873, 71779, 62898, 69962 ，也被上述節點推薦，故幾乎把排名前20名 authority (hub) 的 Node 都(被)推薦到了，因此 Node61398 拿到了最高的 hub 及 authority 值。

- PageRank (參數 d = 0.1)
    - Graph1

```
1 -> 2 -> 3 -> 4 -> 5 -> 6
```

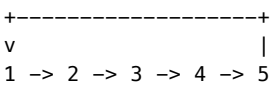
- PageRank



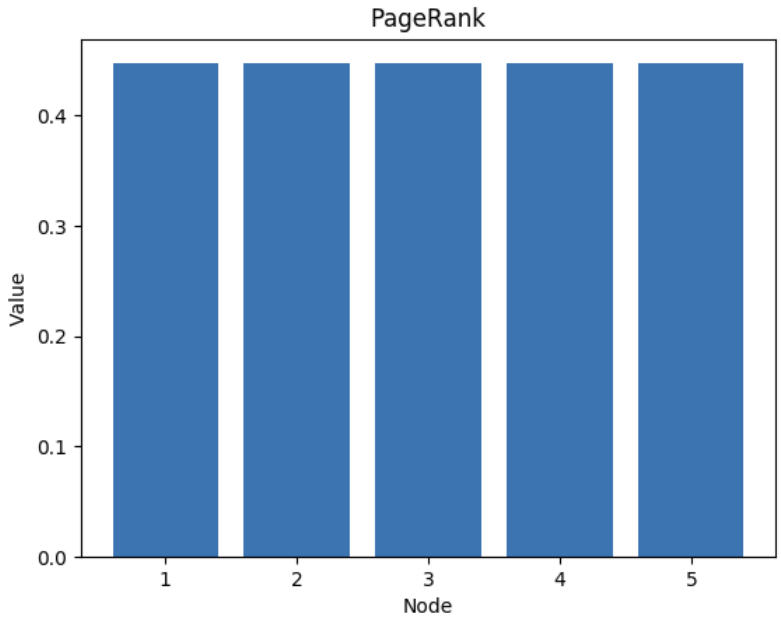
	Node	Value
Top-1	6	0.80986
Top-2	5	0.48352
Top-3	4	0.28121
Top-4	3	0.15580
Top-5	2	0.07806
Top-6	1	0.02987

- Discussion
  - 由於 graph1 是單向往前指的架構，因此坐落在起頭的 Node1 沒有任何的父節點，PageRank只有 d/n 在貢獻， Node2 的話除了 d/n 以外，還可以額外繼承父節點的 PageRank，因此 Node2 PageRank 比起 Node1 更多了，依此類推，愈往後面的節點 PageRank 就愈高。

- Graph2



- PageRank



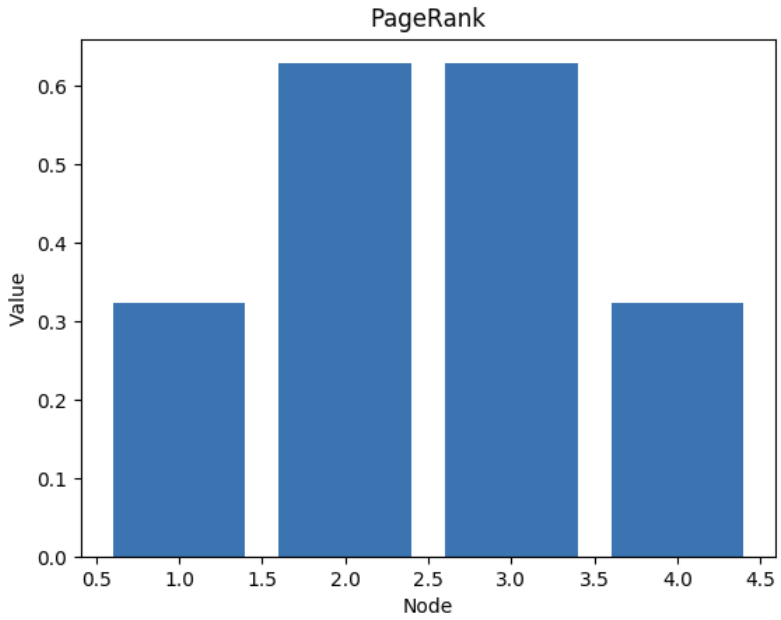
	Node	Value
Top-1	1	0.44721
Top-2	2	0.44721
Top-3	3	0.44721
Top-4	4	0.44721
Top-5	5	0.44721

- Discussion
  - 因為這個 graph 是一個環狀架構，每個節點都是被前一個節點指向，同時也指向下一個節點，沒有任何差異，因此 PageRank 大家都是一致。
  - 如果將此環狀架構打破一個 edge 那麼就會跟 graph1 一樣，愈末端的節點 PageRank 值會愈來愈高。

- Graph3

```
1 <-> 2 <-> 3 <-> 4
```

- PageRank



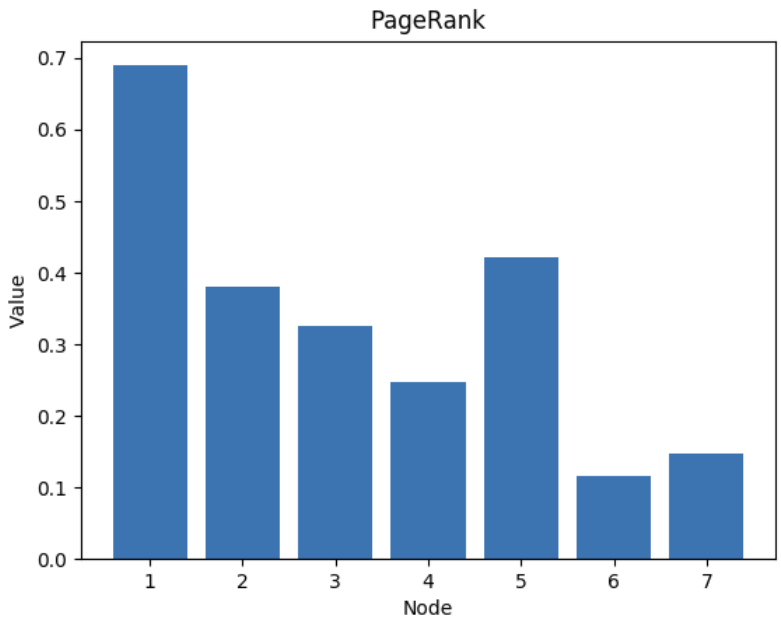
	Node	Value
Top-1	2	0.62885
Top-2	3	0.62885
Top-3	1	0.32334
Top-4	4	0.32334

- Discussion
  - 觀察 graph3 ，若將 Node2 與 Node3 中間之 edge 當分界分為兩個子圖，會發現 Node2 這側的子圖與 Node3 這側的子圖是一模一樣，因此可以預期其 PageRank 會有對稱的情形發生。
  - 對於 Node2 來說，因為 Node1 對外只指向 Node2 ，因此 Node2 可以完全繼承 Node1 的 PageRank ，又能繼承部份 Node3 的 PageRank，但對於 Node1 來說，因為 Node2 對外有連接到 Node1 及 Node3 ，因此只能繼承一半的 Node2 PageRank，故最後的 PageRank 的表現上 Node1 會較 Node2 少。
  - 整體架構因為也是對稱的狀況，Node1 對應上 Node4 及 Node2 對應上 Node3 ，因此 Node1 及 Node4 之 PageRank 互相一致， Node2 及 Node3 之 PageRank 互相一致。

- Graph4

```
Node 1
In  Node: [2, 3, 5, 6]
Out Node: [2, 3, 4, 5, 7]
Node 2
In  Node: [1, 3, 4]
Out Node: [1]
Node 3
In  Node: [1, 4, 5]
Out Node: [1, 2]
Node 4
In  Node: [1, 5]
Out Node: [2, 3, 5]
Node 5
In  Node: [1, 4, 6, 7]
Out Node: [1, 3, 4, 6]
Node 7
In  Node: [1]
Out Node: [5]
Node 6
In  Node: [5]
Out Node: [1, 5]
```

- PageRank

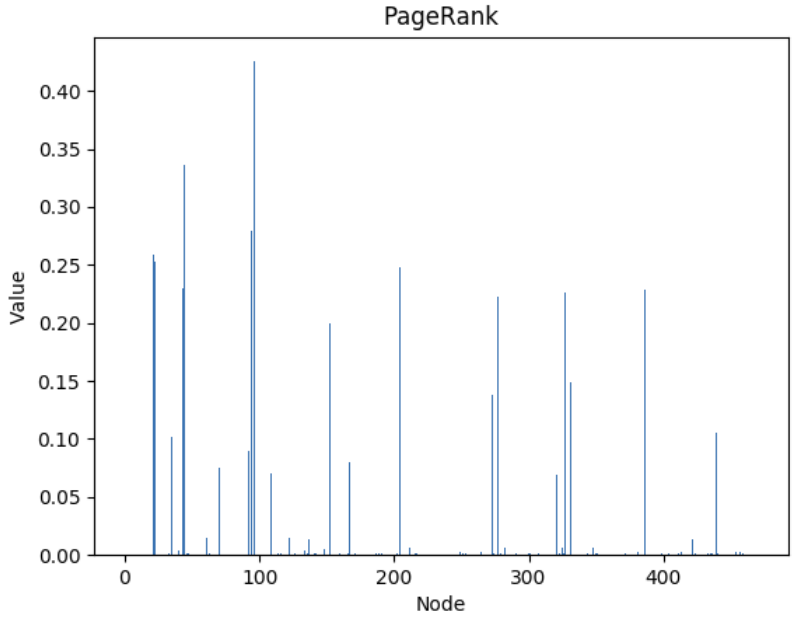


	Node	Value
Top-1	1	0.68973
Top-2	5	0.42070
Top-3	2	0.38094
Top-4	3	0.32583
Top-5	4	0.24719
Top-6	7	0.14681
Top-7	6	0.11553

- Discussion
  - 觀察圖形連接關係，由於 Node1 被 Node2, 3, 5, 6 連接了，而這些連接 Node1 的節點除了 Node5 以外，對外的分支數都在1至2個左右，因此比較能夠承接各個節點完整的 PageRank ，故 Node1 在 PageRank 上的表現勝過其他節點。

- Graph5

- PageRank

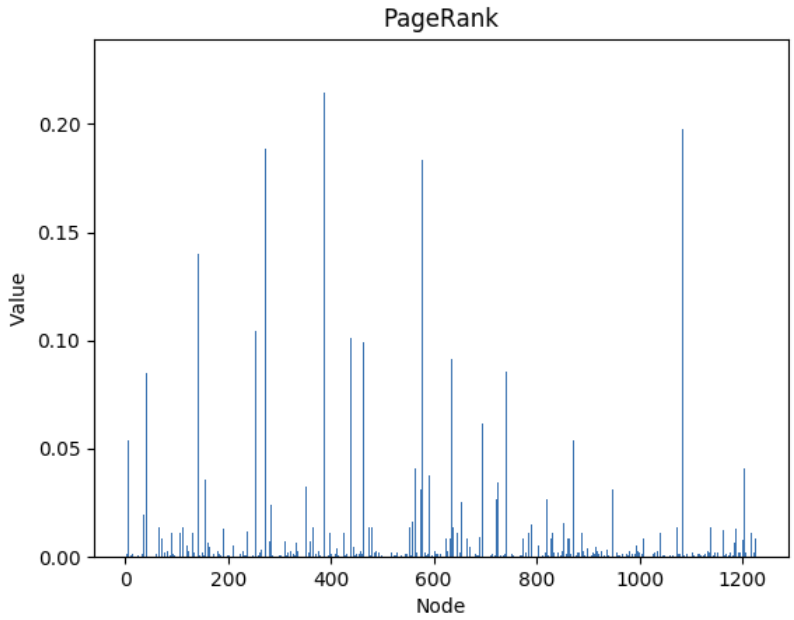


	Node	Value
Top-1	96	0.42530
Top-2	44	0.33663
Top-3	24	0.30515
Top-4	94	0.27994
Top-5	21	0.25852
Top-6	22	0.25318
Top-7	204	0.24792
Top-8	43	0.23022
Top-9	386	0.22897
Top-10	327	0.22667

- Discussion
  - 觀察 graph5 圖， Node96 被 Node43 (out-degree = 8), 70 (out-degree = 9), 94 (out-degree = 8), 109 (out-degree = 6), 152 (out-degree = 9), 204 (out-degree = 8), 273 (out-degree = 8), 277 (out-degree = 8), 321 (out-degree = 8), 327 (out-degree = 9), 369 (out-degree = 8), 386 (out-degree = 9), 439 (out-degree = 7) 連接，以這圖形來說，扣除 out-degree 為 0 的 Node ，平均每個 Node 之 out-degree 為 9.33898 ，又平均每個 Node 之 in-degree 為 2.34968 ，按照 Node96 被連接的狀況，in-degree 高於平均值甚多，且其父親節點的分支數又低於平均值，因此 Node96 在 PageRank 表現上勝過其他節點。

- Graph6

- PageRank



	Node	Value
Top-1	410	0.22794
Top-2	1052	0.22308
Top-3	374	0.21416
Top-4	387	0.21416
Top-5	1021	0.21416
Top-6	554	0.21170
Top-7	43	0.20534
Top-8	415	0.20455
Top-9	468	0.20444
Top-10	494	0.19918

- Discussion
  - 觀察 graph6 圖， Node410 被 Node1 (out-degree = 26), 36 (out-degree = 25), 41 (out-degree = 29), 68 (out-degree = 29), 95 (out-degree = 31), 135 (out-degree = 27), 191 (out-degree = 26), 209 (out-degree = 27), 220 (out-degree = 27), 254 (out-degree = 29), 273 (out-degree = 29), 290 (out-degree = 29), 324 (out-degree = 26), 331 (out-degree = 31), 361 (out-degree = 22), 387 (out-degree = 31), 421 (out-degree = 22), 497 (out-degree = 28), 554 (out-degree = 26), 559 (out-degree = 28), 578 (out-degree = 25), 648 (out-degree = 27), 689 (out-degree = 23), 695 (out-degree = 28), 748 (out-degree = 28), 765 (out-degree = 28), 848 (out-degree = 27), 872 (out-degree = 29), 897 (out-degree = 28), 946 (out-degree = 29), 955 (out-degree = 26), 1058 (out-degree = 30), 1084 (out-degree = 25), 1134 (out-degree = 20), 1139 (out-degree = 28), 1143 (out-degree = 26), 1163 (out-degree = 28) 連接，以這圖形來說，扣除 out-degree 為 0 的 Node ，平均每個 Node 之 out-degree 為 27.91444 ，又平均每個 Node 之 in-degree 為 4.25081 ，按照 Node410 被連接的狀況，in-degree 高於平均值甚多，且其父親節點的分支數接近於平均值，因此 Node410 在 PageRank 表現上勝過其他節點。

- Graph7

> 本資料的構成是由 project1 之 IBM dataset 構成，將每個 transaction 的各個 item 以單向連接到下一個 item，由於總節點數為 7030 ，總邊數為 8487 無法使用圖形呈現，詳細 graph 詳見 `hw3dataset/graph7.txt` 。

- PageRank

	Node	**Value**
Top-1	97823	0.36514
Top-2	88358	0.22892
Top-3	97913	0.21705
Top-4	90065	0.19614
Top-5	99857	0.19566
Top-6	96816	0.17525
Top-7	97286	0.17149
Top-8	97524	0.16362
Top-9	94109	0.14646
Top-10	98458	0.13102
Top-11	89519	0.12745
Top-12	98443	0.12517
Top-13	88436	0.11396
Top-14	93349	0.10772
Top-15	98964	0.10711
Top-16	97657	0.10623
Top-17	91735	0.10573
Top-18	88201	0.10186
Top-19	95487	0.10051
Top-20	90819	0.09970

- Discussion
  - 由 node 及 edge 比例來看，可以得知平均一個節點大約一個至兩個 in-node 及 out-node。
  - Node97823 被 Node90065 (out-degree = 2), 94109 (out-degree = 1) 連接，因此 Node97823 完全承接了 Node94109 (PageRank 第9名) 及一半 Node90065 (PageRank 第4名) 的 PageRank，因此 Node97823 在 PageRank 上的表現勝過其他節點。

◦ Graph8

> 本資料的構成是由 project1 之 IBM dataset 構成，將每個 transaction 的各個 item 以雙向連接到下一個 item，由於總節點數為 7030 ，總邊數為 16974 無法使用圖形呈現，詳細 graph 詳見 `hw3dataset/graph8.txt` 。

- PageRank

	Node	Value
Top-1	24816	0.07222
Top-2	36038	0.06435
Top-3	63977	0.06349
Top-4	52972	0.05563
Top-5	61424	0.05139
Top-6	54931	0.05105
Top-7	28083	0.04689
Top-8	67992	0.04650
Top-9	76922	0.04265
Top-10	39223	0.04261
Top-11	56031	0.04250
Top-12	69839	0.04248
Top-13	49957	0.04244
Top-14	83398	0.04229
Top-15	71418	0.03860
Top-16	32134	0.03837
Top-17	13454	0.03837
Top-18	79288	0.03835
Top-19	22769	0.03834
Top-20	50253	0.03833

        - Discussion
            - 在 graph8 中， Node24816 被 Node22816 (out-degree = 2), 29003 (out-degree = 2), 23531 (out-degree = 6), 38066 (out-degree = 2), 16336 (out-degree = 4), 29643 (out-degree = 4), 22720 (out-degree = 5), 39223 (out-degree = 10), 17387 (out-degree = 2), 25529 (out-degree = 2), 19392 (out-degree = 4), 41690 (out-degree = 4), 34026 (out-degree = 4), 22101 (out-degree = 2), 25016 (out-degree = 2), 18469 (out-degree = 2), 31529 (out-degree = 2) 連接，以整體來說平均每個節點 in-degree 及 out-degree 為 2.41451 ，由 Node24816 之 in-degree 遠大於平均值，且每個父節點大部份分支數都為 2 ，因此在 PageRank 表現上較其他節點好。

- SimRank
    - Graph1

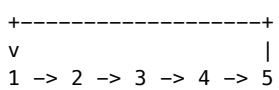
      ```
      1 -> 2 -> 3 -> 4 -> 5 -> 6
      ```

        - SimRank (參數 C = 1.0 及 C = 0.8)

SimRank	Value
SimRank(1, 1)	1.0000
SimRank(1, 2)	0.0000
SimRank(1, 3)	0.0000
SimRank(1, 4)	0.0000
SimRank(1, 5)	0.0000
SimRank(1, 6)	0.0000
SimRank(2, 2)	1.0000
SimRank(2, 3)	0.0000
SimRank(2, 4)	0.0000
SimRank(2, 5)	0.0000
SimRank(2, 6)	0.0000
SimRank(3, 3)	1.0000
SimRank(3, 4)	0.0000
SimRank(3, 5)	0.0000
SimRank(3, 6)	0.0000
SimRank(4, 4)	1.0000
SimRank(4, 5)	0.0000
SimRank(4, 6)	0.0000
SimRank(5, 5)	1.0000
SimRank(5, 6)	0.0000
SimRank(6, 6)	1.0000

- Discussion
  - 參數 C 之設定 1.0 與 0.8 ，結果皆為一樣的。
  - 由於 graph1 是單向往前指的架構，完全不存在有共同父節點的情況 (out-degree 最多只有 1)，因此只有相同節點的 SimRank 為 1 ，其他倆倆組合的節點 SimRank 都為 0 。

- Graph2



- SimRank (參數 C = 1.0 及 C = 0.8)

SimRank	Value
SimRank(1, 1)	1.0000
SimRank(1, 2)	0.0000
SimRank(1, 3)	0.0000
SimRank(1, 4)	0.0000
SimRank(1, 5)	0.0000
SimRank(2, 2)	1.0000
SimRank(2, 3)	0.0000
SimRank(2, 4)	0.0000
SimRank(2, 5)	0.0000
SimRank(3, 3)	1.0000
SimRank(3, 4)	0.0000
SimRank(3, 5)	0.0000
SimRank(4, 4)	1.0000
SimRank(4, 5)	0.0000
SimRank(5, 5)	1.0000

- Discussion
  - 這個 graph 是一個環狀架構，但狀況跟 graph1 一樣，完全不存在有共同父節點的情況 (out-degree 最多只有 1)，因此只有相同節點的 SimRank 為 1 ，其他倆倆組合的節點 SimRank 都為 0 。

- Graph3

```
1 <-> 2 <-> 3 <-> 4
```

- SimRank

SimRank	Value (C = 1.0)
SimRank(1, 1)	1.0000
SimRank(1, 2)	0.0000
SimRank(1, 3)	1.0000
SimRank(1, 4)	0.0000
SimRank(2, 2)	1.0000
SimRank(2, 3)	0.0000
SimRank(2, 4)	1.0000
SimRank(3, 3)	1.0000
SimRank(3, 4)	0.0000
SimRank(4, 4)	1.0000

SimRank	Value (C = 0.8)
SimRank(1, 1)	1.0000
SimRank(1, 2)	0.0000
SimRank(1, 3)	0.6667
SimRank(1, 4)	0.0000
SimRank(2, 2)	1.0000
SimRank(2, 3)	0.0000
SimRank(2, 4)	0.6667
SimRank(3, 3)	1.0000
SimRank(3, 4)	0.0000
SimRank(4, 4)	1.0000

- Discussion
  - 因為這個 graph 的特性，對於最短路徑為基數個 edge 才能到達的節點配對(如(1,2)、(1,4)、(2,3)、(3、4))，其父節點仍然是互相隔了基數個 edge ，故這樣的節點配對，即使不斷的遞迴去找共同的父節點也不可能會找到，因為基數個 edge 才能到達的節點配對其父節點必定隔了基數個 edge 。
  - 而相隔偶數 edge 的節點配對，其父節點也是相隔偶數 (包含相隔 0 個) edge ，因此只要是相隔偶數 edge 的配對不斷遞迴找下去一定都能找到共同父親節點，而在 C = 1 的設定下，可以想像成 SimRank 不會因為遞迴的深度而衰減，因此相隔偶數 edge 的配對其 SimRank 為 1 。
  - 然而我們修改 C = 0.8 時，發現 SimRank(1, 3) 及 SimRank(2, 4) 不再是 1 了，這顯示了當配對節點要從父節點繼承 SimRank 時，需透過折減係數 C 來進行折減，因此繼承而來的 SimRank 會隨著遞迴深度而衰減。

- Graph4

```
Node 1
In  Node: [2, 3, 5, 6]
Out Node: [2, 3, 4, 5, 7]
Node 2
In  Node: [1, 3, 4]
Out Node: [1]
Node 3
In  Node: [1, 4, 5]
Out Node: [1, 2]
Node 4
In  Node: [1, 5]
Out Node: [2, 3, 5]
Node 5
In  Node: [1, 4, 6, 7]
Out Node: [1, 3, 4, 6]
Node 6
In  Node: [5]
Out Node: [1, 5]
Node 7
In  Node: [1]
Out Node: [5]
```

  - SimRank

SimRank	Value (C = 1.0)
SimRank(1, 1)	1.0000
SimRank(1, 2)	1.0000
SimRank(1, 3)	1.0000
SimRank(1, 4)	1.0000
SimRank(1, 5)	1.0000
SimRank(1, 6)	1.0000
SimRank(1, 7)	1.0000
SimRank(2, 2)	1.0000
SimRank(2, 3)	1.0000
SimRank(2, 4)	1.0000
SimRank(2, 5)	1.0000
SimRank(2, 6)	1.0000
SimRank(2, 7)	1.0000
SimRank(3, 3)	1.0000
SimRank(3, 4)	1.0000
SimRank(3, 5)	1.0000
SimRank(3, 6)	1.0000
SimRank(3, 7)	1.0000
SimRank(4, 4)	1.0000
SimRank(4, 5)	1.0000
SimRank(4, 6)	1.0000
SimRank(4, 7)	1.0000
SimRank(5, 5)	1.0000
SimRank(5, 6)	1.0000
SimRank(5, 7)	1.0000
SimRank(6, 6)	1.0000
SimRank(6, 7)	1.0000
SimRank(7, 7)	1.0000

SimRank	Value (C = 0.8)
SimRank(1, 1)	1.0000
SimRank(1, 2)	0.3603
SimRank(1, 3)	0.3490
SimRank(1, 4)	0.3537
SimRank(1, 5)	0.3377
SimRank(1, 6)	0.4151
SimRank(1, 7)	0.2924
SimRank(2, 2)	1.0000
SimRank(2, 3)	0.4068
SimRank(2, 4)	0.3697
SimRank(2, 5)	0.4122
SimRank(2, 6)	0.2854
SimRank(2, 7)	0.4541
SimRank(3, 3)	1.0000
SimRank(3, 4)	0.4496
SimRank(3, 5)	0.3901
SimRank(3, 6)	0.4481
SimRank(3, 7)	0.4510
SimRank(4, 4)	1.0000
SimRank(4, 5)	0.3427
SimRank(4, 6)	0.5351
SimRank(4, 7)	0.5351
SimRank(5, 5)	1.0000
SimRank(5, 6)	0.2731
SimRank(5, 7)	0.4122
SimRank(6, 6)	1.0000
SimRank(6, 7)	0.2701
SimRank(7, 7)	1.0000

- Discussion
  - 我們個別討論每一種配對 SimRank 非 0 的原因，
    - SimRank(1, 2): 透過共同父節點 Node3 。
    - SimRank(1, 3): 透過共同父節點 Node5 。
    - SimRank(1, 4): 透過共同父節點 Node5 。
    - SimRank(1, 5): 透過共同父節點 Node6 。
    - SimRank(1, 6): 透過共同父節點 Node5 。
    - SimRank(1, 7): Node1 取出一個父節點 Node2 ， Node7 取出一個父節點 Node1 ，而依據前面的結論 SimRank(1, 2) 為非 0 。
    - SimRank(2, 3): 透過共同父節點 Node4 。
    - SimRank(2, 4): 透過共同父節點 Node1 。

        - SimRank(2, 5): 透過共同父節點 Node1, 4 。
        - SimRank(2, 6): Node2 取出一個父節點 Node1 ， Node6 取出一個父節點 Node5 ，而依據前面的結論 SimRank(1, 5) 為非 0 。
        - SimRank(2, 7): 透過共同父節點 Node1 。
        - SimRank(3, 4): 透過共同父節點 Node1, 5 。
        - SimRank(3, 5): 透過共同父節點 Node1, 4 。
        - SimRank(3, 6): 透過共同父節點 Node5 。
        - SimRank(3, 7): 透過共同父節點 Node1 。
        - SimRank(4, 5): 透過共同父節點 Node1 。
        - SimRank(4, 6): 透過共同父節點 Node5 。
        - SimRank(4, 7): 透過共同父節點 Node1 。
        - SimRank(5, 6): Node5 取出一個父節點 Node1 ， Node6 取出一個父節點 Node5 ，而依據前面的結論 SimRank(1, 5) 為非 0 。
        - SimRank(5, 7): 透過共同父節點 Node1 。
        - SimRank(6, 7): Node6 取出一個父節點 Node5 ， Node7 取出一個父節點 Node1 ，而依據前面的結論 SimRank(1, 5) 為非 0 。
    - 因此扣除相同節點配對的狀況，依照上面的討論，其他的配對 SimRank 皆不為 0 ，而且還互相糾纏，也就是說自己本身的 SimRank 已經不為 0 的狀況，還可以在繼續接收其父親節點配對產生的 SimRank ，因此這樣的相關性在 C = 1.0 的狀況下 (不會因遞迴深度而損失傳遞的相關性) 最後會收斂到大家都是 1 。
    - 而由 C = 0.8 的數據中我們可以看到有些非零的 SimRank 不再是 1 了，主要是在於遞迴深度使得 SimRank 在繼承父節點配對時產生了衰減，而 SimRank 愈接近 0 者，表示這兩兩配對的節點要找到相關性，必須通過好幾曾的遞迴才能找到共同的父節點。

- Graph5

  > 由於配對數量過多 (約 11 萬種配對)，因此詳細 SimRank 放在 `result/simrank_5_c10.txt` 及 `result/simrank_5_c08.txt` 。

  - Discussion
    - 由結果可以發現，SimRank 分佈開始有很多配對為 0 ，這可能說明著這個 graph 連接的結構沒有辦法讓 SimRank 傳遞到每種配對上，如果把這些 SimRank 為 0 的配對再多個幾條指進來的 edge，應該是會有效的提升 SimRank ，因為這配對可以再透過這些額外連進來的 edge 來去讓其他的父節點配對傳遞非 0 的 SimRank 回來，這樣一來原本為 0 的配對就不會為 0 了。

How to Increase Hub, Authority and PageRank?

- Hub
  - Graph1
    - Origin

      ```
      1 -> 2 -> 3 -> 4 -> 5 -> 6
      ```

    - Modify

      ```
      1 -> 2 -> 3 -> 4 -> 5 -> 6
      |         ^    ^    ^    ^
      |         |    |    |    |
      +----+----+----+----+----+
      ```

    - 我們可以將 Node1 指向其他的節點，使得其他節點的Authority可以分享給 Node1 Hub。
    - 實驗結果 Node1 之 Hub 原為 0.44721 ，提升為 0.92388 。

- Graph2
  - Origin

```
+--------------------+
v                    |
1 -> 2 -> 3 -> 4 -> 5
```

  - Modify

```
+--------------------+
v                    |
1 -> 2 -> 3 -> 4 -> 5
|         ^    ^    ^
+---------+----+----+
```

  - 我們可以將 Node1 指向其他的節點，使得其他節點的Authority可以分享給 Node1 Hub。
  - 實驗結果 Node1 之 Hub 原為 0.44721 ，提升為 0.90958 。
- Graph3
  - Origin

```
1 <-> 2 <-> 3 <-> 4
```

  - Modify

```
    1 <-> 2 <-> 3 <-> 4
    |
+-+-+-+-+
| | | | |
v v v v v
5 6 7 8 9
```

  - 我們可以新增 Node5 ~ Node9 將 Node1 指向新增的節點，便能提升 Node1 之 Hub 值。
  - 實驗結果 Node1 之 Hub 原為 0.60150 ，提升為 0.97325 。

- Authority
  - Graph1
    - Origin

```
1 -> 2 -> 3 -> 4 -> 5 -> 6
```

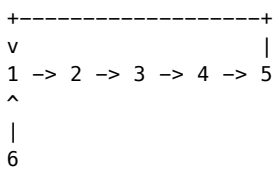
    - Modify

```
1 -> 2 -> 3 -> 4 -> 5 -> 6
^^
| \
7  8
```

    - 我們可以透過新增 Node7 及 Node8 指向 Node1 便能提升 Node1 之 Authority 值。
    - 實驗結果 Node1 之 Authority 原為 0.0 ，提升為 1.0 。
  - Graph2
    - Origin

```
+--------------------+
v                    |
1 -> 2 -> 3 -> 4 -> 5
```

- Modify

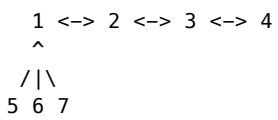


- 我們可以透過新增 Node6 指向 Node1 便能提升 Node1 之 Authority 值。
- 實驗結果 Node1 之 Authority 原為 0.44721 ，提升為 1.0 。

◦ Graph3

- Origin

- Modify

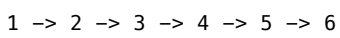


- 我們可以新增 Node5 ~ Node7 將 Node1 指向新增的節點，便能提升 Node1 之 Authority 值。
- 實驗結果 Node1 之 Authority 原為 0.37175 ，提升為 0.92388 。

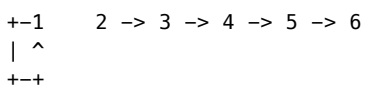
- PageRank

◦ Graph1

- Origin



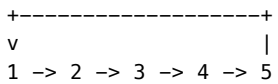
- Modify



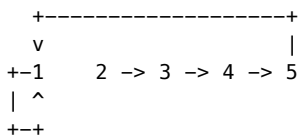
- 我們可以刪除 1->2 之 edge ，新增 1->1 之 edge ，使得 Node1 在 iteration 中 share PageRank 給 child 時只會分配給自己。
- 實驗結果 Node1 之 PageRank 原為 0.02987 ，提升為 0.99137。

◦ Graph2

- Origin



- Modify



- 我們可以刪除 1->2 之 edge ，新增 1->1 之 edge ，使得 Node1 在 iteration 中 share PageRank 給 child 時只會分配給自己。

        - 實驗結果 Node1 之 PageRank 原為 0.44721 ，提升為 0.99492。
    - Graph3
        - Origin

```
1 <-> 2 <-> 3 <-> 4
```

        - Modify

```
+-1 <- 2 <-> 3 <-> 4
| ^
+-+
```

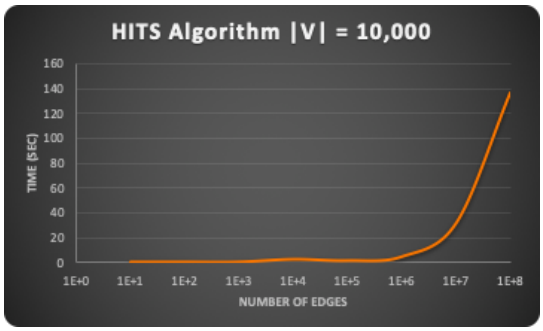
        - 我們可以刪除 1->2 之 edge ，新增 1->1 之 edge ，使得 Node1 在 iteration 中 share PageRank 給 child 時只會分配給自己。
        - 實驗結果 Node1 之 PageRank 原為 0.32334 ，提升為 0.97425 。

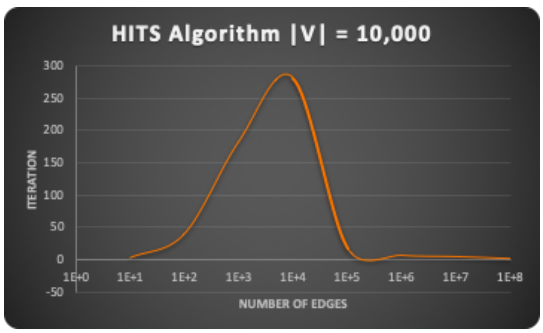
Performance Analysis

以下 graph 皆為 `ggen.py` 產生。

```
$ python ggen.py -v <NUM_VERTICES> -e <NUM_EDGES> -o <FILENAME>
```

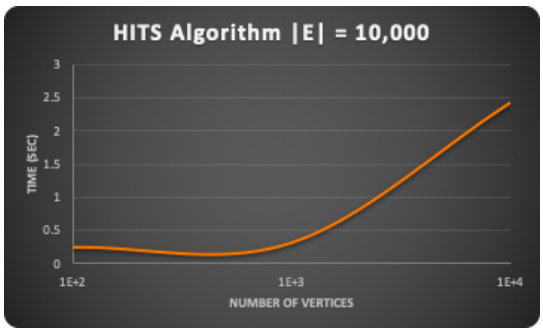
- HITS
    - 固定節點數為 10,000
        - 隨著邊數增加，所需時間如下表

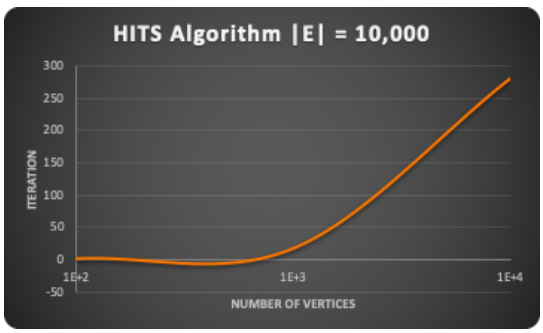




# of edges	time	iteration
10	0m0.222s	2
100	0m0.226s	39
1,000	0m0.277s	182
10,000	0m2.435s	281
100,000	0m1.352s	17
1,000,000	0m4.509s	6
10,000,000	0m31.408s	4
100,000,000	2m16.992s	1

- 固定邊數為 10,000
  - 隨著節點數增加，所需時間如下表





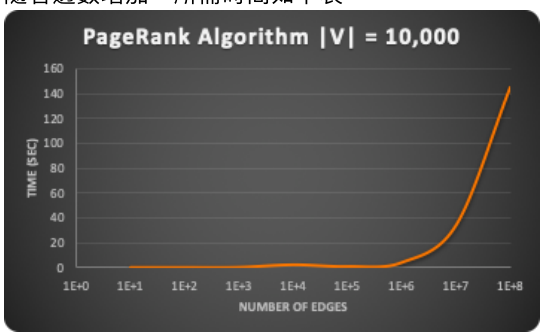
# of vertices	time	iteration
100	0m0.238s	1
1,000	0m0.307s	17
10,000	0m2.435s	281

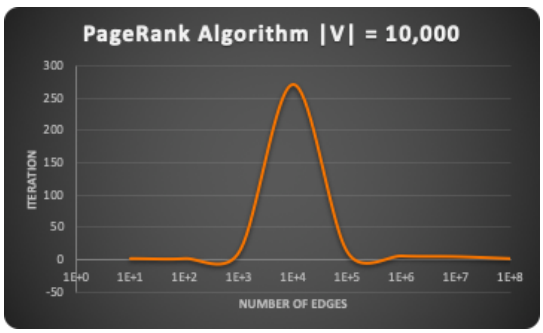
- Discussion
  - 跳出迭代迴圈條件為，更新前後 hub 及 authority 差值的平方和小於 1e-10 。
  - 由圖表可以得知，當固定 vertices 數量時，隨著 edges 數量增加，整體來說時間上會更急速的成長，但是在 |V| = 10,000 且 |E| = 10,000 時我們可以發現，計算時間為 0m2.435s 而， |V| = 10,000 且 |E| = 100,000 時反而計算時間為 0m1.352s ，為了瞭解為什麼在這邊 edges 數上升了，但是計算時間卻變快了這件事，我們又觀察了整體迭代次數，發現在 |V| = 10,000 且 |E| = 10,000 這個狀況下，迭代次數達到最高峰 281 次，因此我麼可以推斷因為迭代次數的暴增，造成 |V| = 10,000 且 |E| = 10,000 這個狀態的執行時間被拉長一點。
  - 然而，要進一步了解為什麼 |V| = 10,000 且 |E| = 10,000 這個狀態的迭代次數暴增，我們觀察 |E| 與 |V| 比值其實就意味著每個節點平均 in-degree 及 out-degree ，當 in-degree 及 out-degree 少量時意味著每次迭代只需要用到幾個父節點或是子節點來更新，甚至是完全沒有 in/out-degree 的節點就是固定 0 不必更新，這就解釋了為什麼剛開始那段比值 |E| 與 |V| 比值較小時，迭代次數不需要太多，但隨著

比值越大，迭代次數就會開始成長，直到 |E| 與 |V| 比值為 1 時達迭代次數最高峰；再來就是後半段隨著 |E| 與 |V| 比值繼續成長，反而迭代次數下降了，其原因應該主要在於隨著 in/out-degree 成長，開始陸陸續續會有些節點的 in/out-degree 已經達到飽和(也就是說跟 graph 其他節點都有互相連接)，最後每個節點 in/out-degree 都達到飽和，在這個時候我們可以預期，每個節點的 hub 及 authority 都會是一樣的(因為每個節點連接的狀況完全一模一樣)，而這樣的預期結果其實就跟我們一開始初始化的 hub 及 authority 是一樣的(一開始 hub 及 authority 的初始化就是設為每個節點都相同)，因此也不用迭代太多次就找到解了。

- 在固定 edges 數的圖表部分，也能驗證上述結論，在 |E| 與 |V| 比值越來越接近 1 時，除了計算時間增加外，其迭代次數越來越高。而實驗只做到 vertices 數為 10,000 ，其原因是如果以一個 100,000 節點(往上一個層級)的圖來說，總共有 1e+10 個 edges 要選擇，而 int 類型的 random 無法提供這麼大的數字隨機，造成 error ，因此 vertices 數量我們只實驗到 10,000。
- 最後，我們分析每一次迴圈內的時間複雜度，因為每一輪的更新都需要看過兩次(authority 及 hub)每個 edge ，而每個 vertex 都至少看過一次，因此一次迭代所需計算時間其時間複雜度為 O(|V| + |E|) 。

- PageRank
  - 固定節點數為 10,000
    - 隨著邊數增加，所需時間如下表

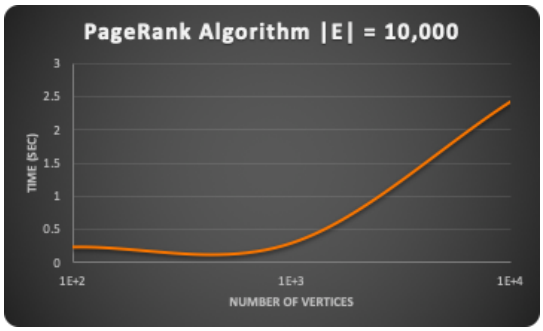


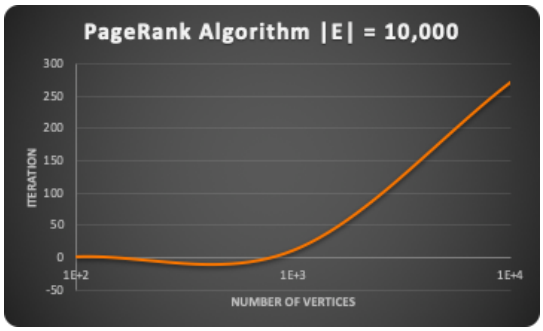


# of edges	time	iteration
10	0m0.225s	1
100	0m0.240s	1
1,000	0m0.387s	11
10,000	0m2.430s	271
100,000	0m0.998s	11
1,000,000	0m4.235s	5
10,000,000	0m34.173s	4
100,000,000	2m25.467s	1

  - 固定邊數為 10,000

- 隨著節點數增加，所需時間如下表

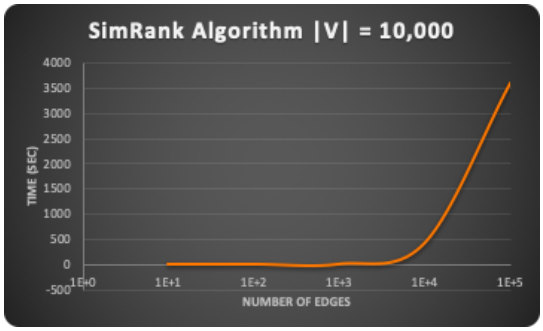


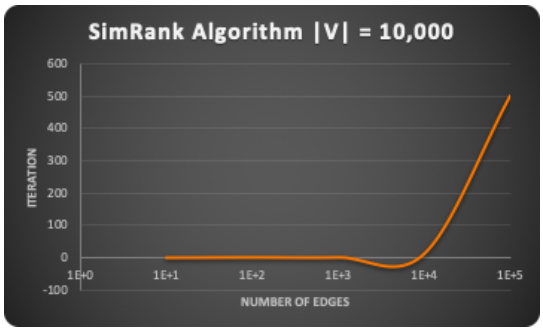


# of vertices	time	iteration
100	0m0.236s	1
1,000	0m0.295s	11
10,000	0m2.430s	271

  - Discussion
    - 跳出迭代迴圈條件為，更新前後 PageRank 差值的平方和小於 1e-10 。
    - 由於 PageRank 計算的方式跟 HITS 類似，一樣要經歷多次迭代，而每一次的迭代都需要掃描全部的節點，對於全部節點都採取其父節點 PageRank 來更新，因此每個節點的計算量大約會與其 in-degree 成正比。
    - 因此我們可以在圖表上看到了不論是計算時間或迭代次數都有與 HITS 類似的曲線，以計算時間來說整體會隨著節點數或邊數增加，計算時間也會跟著增加，而迭帶次數會隨著 |E| / |V| 值愈接近 1 而會有愈來愈高的趨勢，愈不接近 1 的話而會有愈來愈低的趨勢。
    - 而在固定 vertices 增加 edges 的實驗中也可以看到計算時間在 |V| = 10,000 且 |E| = 10,000 的狀況下會有微微的隆起現象，我想這也是由於在這個狀況下迭代次數暴增，而使得運算時間被拉長一點的緣故。
    - 最後，我們分析每一次迴圈內的時間複雜度，由於計算方式是類似的，因此可以預期每個迴圈的時間複雜度也是一樣的，在每一輪的更新中都需要看過一次每個 edge ，而每個 vertex 都至少看過一次，因此一次迭代所需計算時間其時間複雜度為 O(|V| + |E|) 。

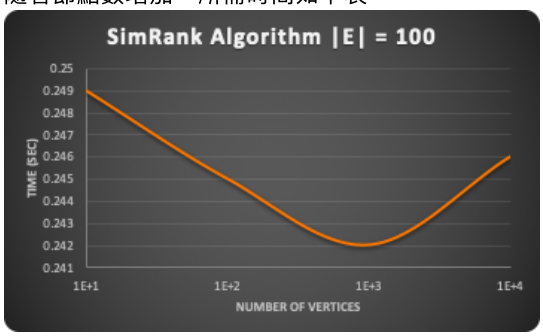
- SimRank
  - 固定節點數為 10,000
    - 隨著邊數增加，所需時間如下表

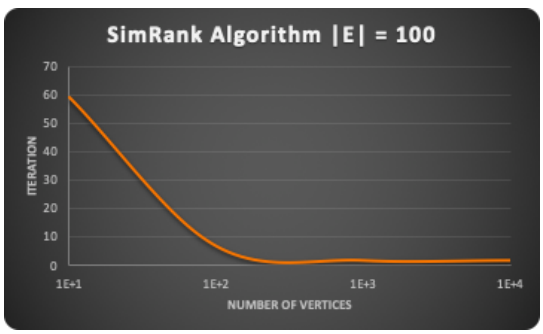




# of edges	time (TLE: > 60 min)	iteration
10	0m0.222s	1
100	0m0.237s	2
1,000	0m2.549s	2
10,000	7m1.164s	13
100,000	TLE	TLE

- 固定邊數為 100
  - 隨著節點數增加，所需時間如下表





# of vertices	time	iteration
10	0m0.249s	59
100	0m0.245s	7
1,000	0m0.242s	2
10,000	0m0.246s	2

- Discussion
  - 跳出迭代迴圈條件為，更新前後 SimRank 差值的平方和小於 1e-10 。
  - 我們分析每次迴圈所需時間複雜度，在每次迴圈中我們必須更新 |V| 中任選 2 vertices 配對之 SimRank ，每個配對更新都需要看過各別 in-degree 乘積個父節點，因此計算量為$\Sigma_{v_1,v_2 \in V}(|Parent(v_1)| \times |Parent(v_2)|)$，整理一下這個式子可以得到$\Sigma_{v_1 \in V}(|Parent(v_1)| \times (|E| - |Parent(v_1)|))$，因此每次迴圈的時間複雜度約為 O(|V|^2 * |E|) 。

    - 由上面時間複雜度的分析可以瞭解 SimRank 時間成長遠高於 HITS 及 PageRank ，因此在做固定 |V| = 10,000 實驗中， |E| = 100,000 跑了好一段時間都無法收斂，而在固定 |E| 的實驗中，也沒辦法順利跑完 |E| = 10,000 各項實驗，因此改由負擔較小的 |E| = 100。
    - 由實驗數據看的出來，隨著 |E| 與 |V| 之比值增加，迭代次數不斷的增加，與 HITS 及 PageRank 不同的點在於迭代次數的高峰不是出現在 |E| / |V| = 1 之處，而是出現於 |E| / |V| 最大值時 (in-degree 及 out-degree 飽和時) ，其原因在於如果每個節點互相緊密相連的話，其最後 SimRank 結果必定為全部 1.0 ，而我們在計算 SimRank 時初始化值除了自己跟自己配對的組合外，其他節點配對皆為 0.0 ，在收斂過程中每種配對都要從 0 不斷的成長到 1 ，這過程會因為互相糾纏的父節點數量過多，而導致 SimRank 成長愈緩慢，越難收斂，最終使得迴圈次數愈來愈多次才能收斂。
    - 在固定 |V| = 10,000 實驗中，我們可以看到，迭代次數的增加搭配上每單位迭代需要計算量增加，使得整體計算時間暴增：另外在固定 |E| = 100 實驗中，雖然每次迭代計算時間會隨著 |V| 增加而成長，但因為 |V| 愈大，意味著 |E| / |V| 愈小，也就反應的迭代次數愈少，因此在這實驗中整體所需花費時間才會約略相等。

Discussion

- **More limitations about link analysis algorithms**
    - 對於基本的 graph1~6 圖形，即使使用更新前後差值平方和小於 1e-10 為跳出迴圈條件，也能夠在幾毫秒計算時間內完成收斂，但在效能分析時，使用隨機產生圖形方式產生了大量節點及邊的圖形，迭代次數就會開始影響到整體的運算時間，依照演算法的特性 HITS 及 PageRank 的迭代次數高峰會在於 |E| / |V| 值為 1 時， SimRank 迭代次數高峰會在於 |E| / |V| = |V| 時 (最大值) ，因此我們可以讓這些需要大量迭代才能收斂的案例，為增快結果的輸出，我們可以動態的看 |E| / |V| 來決定迭代次數，例如 HITS 及 PageRank 我們可以設定如果 |E| / |V| 愈接近 1 時，跳出迴圈的條件較為寬鬆點，或是給一個可以接受的最大迴圈數，相對的， SimRank 我們可以設定 |E| / |V| 愈接近 |V| 時採取一樣的策略，用一點精度的損失來換取計算速度。
- **Can link analysis algorithms really find the “important” pages from Web?**
    - 依照前面採取幾個方法可以增加 authority 、 hub 及 PageRank 的實驗上，真正要找到真正重要的網站應該不是那麼容易。如果以 authority 來衡量網站重要程度的話，我們可以大量架設網站，全部指向要提升 authority 的那個網站，如此一來就能得到高 authority ；如果要以 PageRank 來衡量一個網站重要程度的話，對於要提升 PageRank 的網站，我們可以斷開這網站的 out-degree ，同時在架設多個網站單一指向這個要提升 PageRank 的網站，如此一來 PageRank 就能得到提升。
    - 上述的策略都能夠提升所謂衡量網站重要的指標，但是實際上這網站並不會因為採取這些策略而真正的重要程度有所提升。
- **What are practical issues when implement these algorithms in a real Web?**
    - 我想這要應用到實際上網站評分會面臨最關鍵的問題在於計算時間，從前面的效能分析就能看的出來，這樣的分析所需計算時間會隨著節點數及邊數的提升而有快速成長的趨勢，而真實的搜尋引擎要處理的網站節點量可想而知是超出我們實驗範圍很多的，像是 google 隨便搜尋一個辭彙都會在 0.5 秒鐘內找出 5,000 多萬以上個符合網站，如此才能提供使用者一個可接受的搜尋，但我們實驗中 |V| = 10,000, |E| = 10,000,000 就要跑個半分鐘以上，實在是很難被使用者所接受，因此要把這樣的方法應用到實際的運作上存在難以克服的難度。
- **Any new idea about the link analysis algorithm?**
    - 由於 link analysis algorithm 面臨最大的問題在於運算時間需要很大的負擔，在面臨真實網站處理時，這麼漫長的處理時間是不被允許的，或許我們可以先行在 off-line 依據 similarity 進行網站的分群，把相似度高的網站組合成群，如此一來搜尋引擎在搜尋網站時可以直接計算每個大群的 PageRank 或是 Authority ，在近一步到這些評分較高的網站群內，再做群內部的 PageRank 或是 Authority 計算工作，這或許會舒緩龐大的運算量。
- **What is the effect of “C” parameter in SimRank?**
    - 承如前面的 result 呈現，在兩兩配對的節點計算 SimRank 時，會因為兩節點想透過遞迴去找尋到相同的父節點，但每通過一層的遞迴所繼承而來的 SimRank 都會被 C 值所折減，因此隨著 C 值的設定愈接近 0 的話，兩兩不同的節點互相配對其 SimRank 值必定

會愈接近 0 ，而 C 值愈接近 1 的話，兩兩不同的節點互相配對，在追溯到共同父節點時，其繼承的 SimRank 將不會有折減。

- **Design a new link-based similarity measurement**
  - 或許我們可以考慮以**共同父節點接近程度**來衡量點跟點之間的相似度；為了實現前述目的，我們使用 Floyd-Warshall Algorithm 找出每兩兩配對的節點最短距離，因此我們就能夠知道節點距離關係，然而我們根據節點 i 到達節點 j 的最短距離 d_{ij}，訂出向量 $V_i = [e_{i0}, e_{i1}, \ldots, e_{ij}, \ldots]$ 其中 $e_{ij} = C^{d_{ij}}$ (C 為 0~1 之間的數)，而這個向量意味著節點 i 到 j 接近的程度，接著我們把這向量正規化，最後如果要計算 a 與 b 點的 similarity 我們可以把正規化後的 Va 及 Vb 進行內積運算，便可知道**共同父節點接近程度**。
  - 前述方法已實作在 `main.py` ，以下展示原本方法與上述方法對於 `graph_4.txt` 的差異。

Pairs	SimRank (C=0.8)	NewSimRank (C=0.2)
(1, 1)	1.0000	1.0000
(1, 2)	0.3603	0.4062
(1, 3)	0.3490	0.4071
(1, 4)	0.3537	0.2719
(1, 5)	0.3377	0.4058
(1, 6)	0.4151	0.2598
(1, 7)	0.2924	0.2468
(2, 2)	1.0000	1.0000
(2, 3)	0.4068	0.2935
(2, 4)	0.3697	0.2697
(2, 5)	0.4122	0.1554
(2, 6)	0.2854	0.0682
(2, 7)	0.4541	0.0930
(3, 3)	1.0000	1.0000
(3, 4)	0.4496	0.2938
(3, 5)	0.3901	0.2951
(3, 6)	0.4481	0.0977
(3, 7)	0.4510	0.1224
(4, 4)	1.0000	1.0000
(4, 5)	0.3427	0.4085
(4, 6)	0.5351	0.1222
(4, 7)	0.5351	0.0945
(5, 5)	1.0000	1.0000
(5, 6)	0.2731	0.3852
(5, 7)	0.4122	0.2654
(6, 6)	1.0000	1.0000
(6, 7)	0.2701	0.0927
(7, 7)	1.0000	1.0000

- **What you learned from this project and your comments about this project**
  - 雖然我們實作出來的演算法計算速度遠遠不足實際上搜尋引擎的要求，但由這個 project 我們還是能瞭解到搜尋引擎的運作原理，以及可能透過一些手段提升這些演算法分析出來的 rank ，因此我們在搜尋引擎所找到的網站也未必是真正重要的，或許這些網站只是用了某些手段提升了 rank 使得搜尋引擎比較推薦而已。

Authors

Yu-Tong Shen (https://github.com/yutongshen/)

tags: `Data Mining`