DATA MINING
[CSIE 7111] FALL 2022
PROJECT 3

Student Name: 曾中柏
Student ID: P76101623

Table of Contents

PART 1 FILE STRUCTURE

```
P76101623_DMProject3
├── results
│   ├── graph_1
│   │   ├── graph_1_HITS_authority.txt
│   │   ├── graph_1_HITS_hub.txt
│   │   ├── graph_1_PageRank.txt
│   │   └── graph_1_SimRank.txt
│   ├── graph_2
│   │   ├── graph_2_HITS_authority.txt
│   │   ├── graph_2_HITS_hub.txt
│   │   ├── graph_2_PageRank.txt
│   │   └── graph_2_SimRank.txt
│   ├── graph_3
│   │   ├── graph_3_HITS_authority.txt
│   │   ├── graph_3_HITS_hub.txt
│   │   ├── graph_3_PageRank.txt
│   │   └── graph_3_SimRank.txt
│   ├── graph_4
│   │   ├── graph_4_HITS_authority.txt
│   │   ├── graph_4_HITS_hub.txt
│   │   ├── graph_4_PageRank.txt
│   │   └── graph_4_SimRank.txt
│   ├── graph_5
│   │   ├── graph_5_HITS_authority.txt
│   │   ├── graph_5_HITS_hub.txt
│   │   ├── graph_5_PageRank.txt
│   │   └── graph_5_SimRank.txt
│   ├── graph_6
│   │   ├── graph_6_HITS_authority.txt
│   │   ├── graph_6_HITS_hub.txt
│   │   ├── graph_6_PageRank.txt
│   │   └── graph_6_SimRank.txt
│   └── ibm-5000
│       ├── ibm-5000_HITS_authority.txt
│       ├── ibm-5000_HITS_hub.txt
│       ├── ibm-5000_PageRank.txt
│       └── ibm-5000_SimRank.txt
├── src
│   ├── main.py
│   ├── read_write.py
│   ├── hits.py
│   ├── pagerank.py
│   └── simrank.py
└── report.pdf
```

Figure 1. File structure of this project

PART 2 FIND A WAY TO INCREASE HUB, AUTHORITY, AND PAGERANK OF NODE 1 IN THE FIRST 3 GRAPHS

2.1 Graph 1

由於 Authority 的計算是將 Parent 的 Hub 加總，而 Hub 的計算方式是將 Children 的 Authority 加總，而 PageRank 跟連進來的 In-Degree 有關。所以為了增加 Node 1 的 Authority、Hub、以及 PageRank，我採用的方法是將 Node 1 都連到每一個 Node，且每個 Node 也都會連到 Node 1 (如 Figure 2)，結果如 Table 1 和 Table 2 所示。

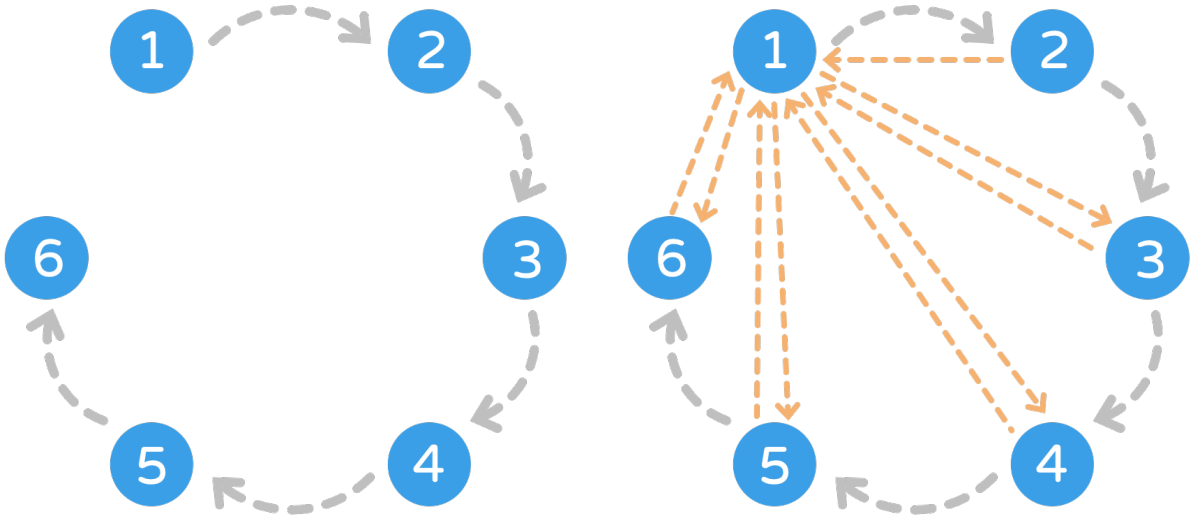

Figure 2. 左圖為 Graph 1，右圖為新加 Edge (橘色 Edge)後的圖

Table 1. Graph 1 的 Authority、Hub、以及 PageRank

	Node 1	Node 2	Node 3	Node 4	Node 5	Node 6
Authority	0	0.2	0.2	0.2	0.2	0.2
Hub	0.2	0.2	0.2	0.2	0.2	0
PageRank	0.056086	0.106564	0.151994	0.192881	0.229679	0.262797

Table 2. 更改後的 Graph 1 的 Authority、Hub、以及 PageRank

	Node 1	Node 2	Node 3	Node 4	Node 5	Node 6
Authority	0.269599	0.099506	0.157724	0.157724	0.157724	0.157724
Hub	0.269599	0.157724	0.157724	0.157724	0.157724	0.099506
PageRank	0.36773	0.082858	0.120144	0.136923	0.144473	0.147871

結果與討論：

- 從 Table 2 可看到，Node 1 的 Authority、Hub、以及 PageRank 都有上升。
- Authority 會增加是因為 Node 1 的 Parent 數量增加了，得到的 Parent 的 Hub 的加總也因此增加。
- Hub 會增加也是因為 Children 增加，所以 Children 的 Authority 的加總也增加了。
- 而 Node 1 的 In-Degree 最高，導致 PageRank 的傳遞大部分最後都會流到 Node 1，所以 Node 1 的 PageRank 最高。

2.2 Graph 2

我用同樣的方法處理 Graph 2，讓 Node 1 連到全部的 Node，且全部的 Node 也都連到 Node 1 (如 Figure 3)，結果如 Table 3 和 Table 4 所示。

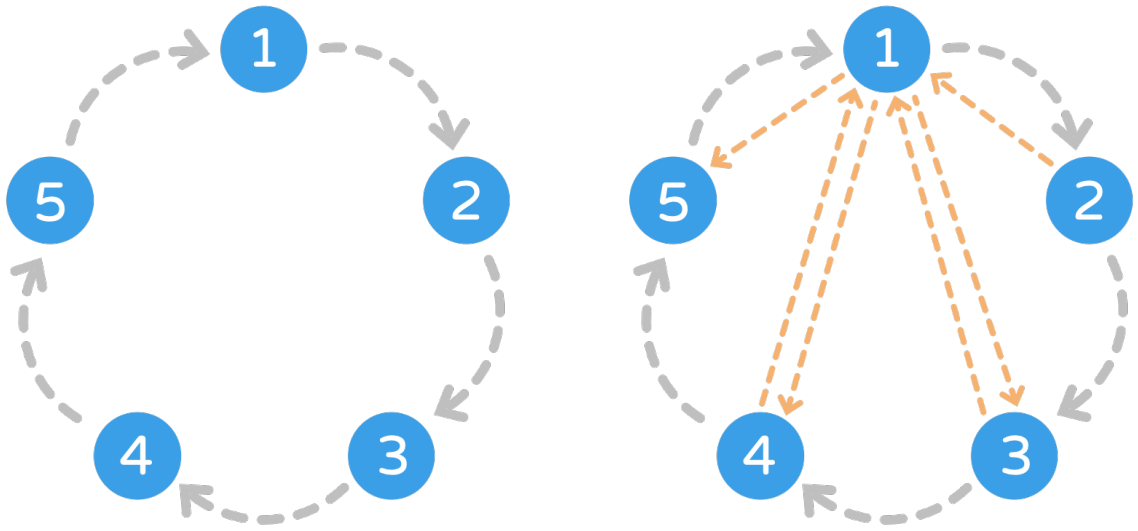

Figure 3. 左圖為 Graph 2，右圖為新加 Edge (橘色 Edge)後的圖

Table 3. Graph 2 的 Authority、Hub、以及 PageRank

	Node 1	Node 2	Node 3	Node 4	Node 5
Authority	0.2	0.2	0.2	0.2	0.2
Hub	0.2	0.2	0.2	0.2	0.2
PageRank	0.2	0.2	0.2	0.2	0.2

Table 4. 更改後的 Graph 2 的 Authority、Hub、以及 PageRank

	Node 1	Node 2	Node 3	Node 4	Node 5
Authority	0.288977	0.117445	0.197859	0.197859	0.197859
Hub	0.288977	0.197859	0.197859	0.197859	0.117445
PageRank	0.381397	0.105814	0.153431	0.174858	0.1845

結果與討論：

- 從 Table 3 和 Table 4 可以看到，用這個方法同樣也可以讓 Node 1 的 Authority、Hub、以及 PageRank 上升，且是全部的 Node 中最高的。

2.3 Graph 3

Graph 3 也用前面的方法處理，如 Figure 4，結果則如 Table 5 和 Table 6 所示。

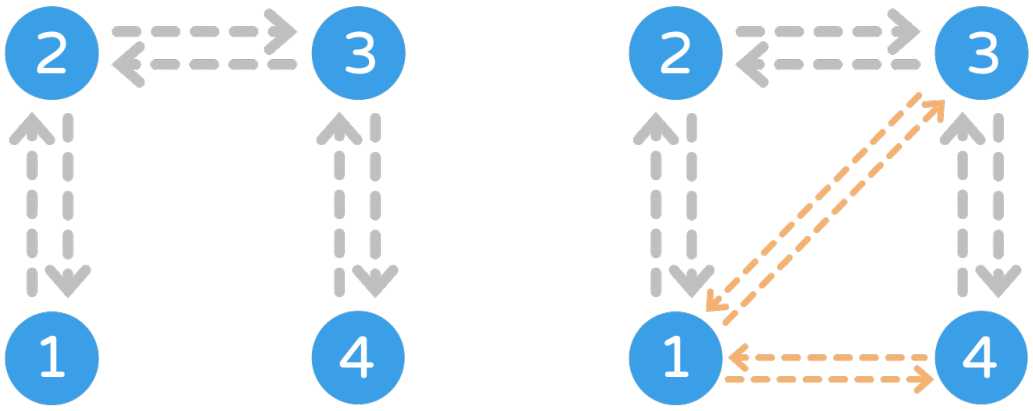

Figure 4. 左圖為 Graph 3，右圖為新加 Edge (橘色 Edge)後的圖

Table 5. Graph 3 的 Authority、Hub、以及 PageRank

	Node 1	Node 2	Node 3	Node 4
Authority	0.190983	0.309017	0.309017	0.190983
Hub	0.190983	0.309017	0.309017	0.190983
PageRank	0.172414	0.327586	0.327586	0.172414

Table 6. 更改後的 Graph 3 的 Authority、Hub、以及 PageRank

	Node 1	Node 2	Node 3	Node 4
Authority	0.280776	0.219224	0.280776	0.219224
Hub	0.280776	0.219224	0.280776	0.219224
PageRank	0.296875	0.203125	0.296875	0.203125

結果與討論：

- 從 Table 5 和 Table 6 的結果可以看到，這個方法同樣可以提升 Node 1 的 Authority、Hub、和 PageRank。
- 此外，也可以看到由於 Node 1 和 Node 3 的 In-Degree 和 Out-Degree 都相同，所以這兩個 Node 的 Authority、Hub、以及 PageRank 都相同。Node 2 和 Node 4 也是如此。

PART 3 ALGORITHM DESCRIPTION

3.1 HITS

HITS 演算法主要是藉由不停更新每個網頁的 Authority 和 Hub，最後會達到收斂，這演算法也就結束。計算 Authority，必須要知道有多少跟 Query 有關的網頁連到我目前的這個網頁；Hub 則是我目前的網頁，有連到多少跟這個 Query 有關的網頁。實際的 Authority 和 Hub 計算方式如下：

$$Authority(v) = \sum_{w \in parent(v)} Hub(w)$$

$$Hub(v) = \sum_{w \in children(v)} Authority(w)$$

從公式可以看到，某一個 Vertex v 的 Authority，是來自於他的 Parent 的 Hub 的加總；Hub 則是 Vertex v 連出去的 Children 的 Authority 的加總。

以下是我的程式碼說明：

假設已經將 Graph 的 Adjacency Matrix 建立起來，並儲存在 `adj_matrix` 這個變數，而 `vertex_size` 則是紀錄總共有多少 Vertex。

首先，將每個 Vertex 的 Authority 和 Hub 都初始化成 1：

```
# 初始化 hub 和 authority，初始值都設為 1
hub = [1 for x in range(vertex_size)]
authority = [1 for x in range(vertex_size)]
```

再來則是進入 while 迴圈，不停更新每個 Vertex 的 Authority 和 Hub，直到達到設定的終止條件，詳細說明如程式碼中的註解：

```
iteration = 1
while True:

    # 將前一個 iteration 計算得到的 hub 和 authority 先 copy 一份
    prev_hub = hub.copy()
    prev_authority = authority.copy()

    # 產生 adj_matrix 的 transpose
    transpose = [[adj_matrix[j][i] for j in range(len(adj_matrix))] for i in
range(len(adj_matrix[0]))]
```

```
    # 用前一個 iteration 的 hub 來計算 authority
    authority = np.dot(transpose, prev_hub)
    authority = authority.astype(float)

    # 用前一個 iteration 的 authority 來計算 hub
    hub = np.dot(adj_matrix, prev_authority)
    hub = hub.astype(float)

    # 記得要做 Normalization，這樣才會收斂
    authority = one_norm(authority, vertex_size)
    hub = one_norm(hub, vertex_size)

    # 計算前一輪跟後一輪的差值
    diff = 0
    for i in range(vertex_size):
        diff += abs(authority[i] - prev_authority[i])
        diff += abs(hub[i] - prev_hub[i])

    # 判斷是否達到 iteration 上限，或前一輪與後一輪差值是否達到設定的收斂門檻 epsilon
    # 若達到上限或門檻，則演算法結束
    if diff < epsilon or iteration > max_iteration:
        break
    else:
        iteration += 1
```

3.2 PageRank

PageRank 公式：

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

PageRank 演算法，根據公式，就是將每個 Vertex 的 Parent 的 PageRank 除以該 Parent 的 Children 數量(也就是該 Parent 的 Out-degree)，將每個 Parent 的加總起來，即為該 Vertex 的 PageRank。另外，為了避免 Rank Sink Problem (詳細內容後面會說明)，多加了 Damping Factor，讓每一個 Vertex，他是有一定的機率是隨機到達該 Vertex，也有一定的機率是從該 Vertex 的 Parent 到達該 Vertex。

以下是我的程式碼說明：

首先，將每個 Vertex 的 PageRank 都初始化成 1/(vertex_size)：

```
# 先初始化每個 vertex 的 PageRank 為(1/vertex_size)
page_rank = [(1/vertex_size) for x in range(vertex_size)]
```

將每個 Vertex 的 Out-degree 記錄起來，方便後面使用：

```
# 記錄每個 vertex 的 out-degree
```

```
outdegree_list = []
for i in range(vertex_size):
    sum = 0
    for j in range(vertex_size):
        sum += adj_matrix[i][j]
    outdegree_list.append(sum)
```

將每個 Vertex 的 Parent 記錄起來，方便後面使用：

```
# 記錄每個 node 的 parent 有誰
parent_list = []
for i in range(vertex_size):
    # 開始尋找 node i 的 parent 有誰
    parent = []
    for j in range(vertex_size):
        if adj_matrix[j][i] == 1:
            # 若有 node i 的 parent，就把它存進 parent
            parent.append(j)

    if len(parent) == 0:
        # 若該 node 沒有 parent，則存[-1]
        parent_list.append([-1])
    else:
        parent_list.append(parent)
```

進入 while 迴圈，每一個 Iteration 都會將每一個 Vertex 的每個 PageRank 都更新一次，詳細說明如程式碼中的註解：

```
iteration = 1
while True:

    # 先將前一個 iteration 的 PageRank copy 一份
    prev_page_rank = page_rank.copy()

    for i in range(vertex_size):
    # 先來更新第 i 個 node 的 page rank
        new_page_rank_sum = 0

        # 尋找第 i 個 node 的 parent
        for j in parent_list[i]:
            if j != -1:
                # 若 i 的 parent j 不等於-1，就代表 node i 有 parent
                # 將 parent 前一個 iteration 的 PageRank 除以他的 outdegree
                # 並加入 new_page_rank_sum
                new_page_rank_sum += (prev_page_rank[j]/outdegree_list[j])

        # 帶入公式更新 node i 的 PageRank
        page_rank[i] = ((damping_factor/vertex_size) +
                        (1-damping_factor) * new_page_rank_sum)

    # 每回合都要將 PageRank 做正規化
    page_rank = one_norm(page_rank, vertex_size)
```

```
# 僅依照 max iterattion 決定是否 break
if iteration >= max_iteration:
    break
else:
    iteration += 1
```

3.3 SimRank

SimRank 公式：

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

SimRank 主要是計算兩個 Vertex，a 和 b，他們之間的相似度 $S(a,b)$。$S(a,b)$的計算方式是將 a 和 b 的 Parent，兩兩將其 Parent 的相似度加總起來，除以 a 和 b 的 Parent 數量的相乘，再乘上 Decay Factor C。

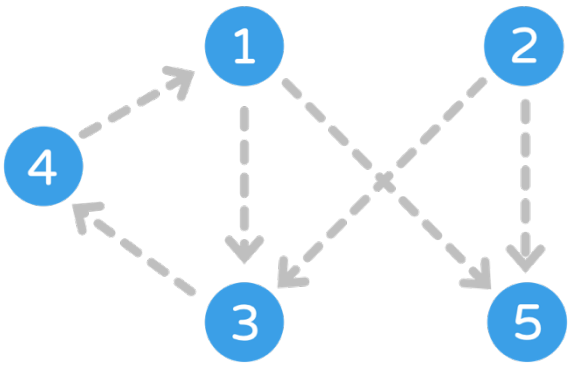

Figure 5. SimRank 說明範例

以 Figure 5 做說明，假設要計算$S(3,5)$：

- 3 的 Parent 有兩個，分別是 1 和 2
- 5 的 Parent 有兩個，分別也是 1 和 2
- 接下來要兩兩將其 Parent 的相似度加總起來：
  所以要將$S(1,1)$、$S(1,2)$、$S(2,1)$、以及$S(2,2)$加總起來
  - 由於自己跟自己的相似度是 1，所以$S(1,1) = S(2,2) = 1$
  - 由於目前在第一個 Iteration，所以$S(1,2) = S(2,1) = 0$
- 所以$S(3,5)$經由第一個 Iteration 計算得到的結果就是：

$$S(3,5) = decay\ factor \times \frac{1+1+0+0}{2\times 2} = decay\ factor \times \frac{2}{4}$$

再以$S(4,5)$為例：

- 4 的 Parent 有 3
- 5 的 Parent 有 1 和 2
- 將 4 和 5 的 Parent 的相似度加總起來：
  由於是第一個 Iteration，所以 $S(3,1)+S(3,2)=0+0=0$
- 所以$S(4,5)$經由第一個 Iteration 計算得到的結果就是：

$$S(4,5)=decay\ factor\ \times\frac{0}{1\times 2}=0$$

第一個 Iteration 將每個 Vertex 跟每個 Vertex 的 Similarity 都算完後，就再進入第二個 Iteration 做計算，直到達到設定的 max_iteration，或收斂為止。

以下是我的程式碼說明：

先宣告一個大小為 vertex_size × vertex_size 的矩陣 Sim 來記錄 Similarity，並將每一格都初始化成 0：

```
# 先初始化 sim
sim = [[0 for x in range(vertex_size)] for y in range(vertex_size)]
```

由於每個 Vertex 對自己的 Similarity 是 1，因此先將 Sim 的對角線都初始化成 1：

```
# 將 sim 的對角線初始化成 1 (∵ Sim(i,i)=1)
for i in range(vertex_size):
    sim[i][i] = 1
```

將每個 Vertex 有多少 In-Neighbor、以及有哪些 In-Neighbor 記錄下來，方便後面使用：

```
# 記錄每個點的 in-neighbor 數量
in_neighbors_num = []
for i in range(vertex_size):
    sum = 0
    for j in range(vertex_size):
        sum += adj_matrix[j][i]
    in_neighbors_num.append(sum)

# 記錄每個點的 in-neighbor 有誰
in_neighbor = []
for i in range(vertex_size):
    parent = []
    for j in range(vertex_size):
        if adj_matrix[j][i] == 1:
            parent.append(j)
    in_neighbor.append(parent)
```

接下來進入 while 迴圈執行 SimRank，詳細說明如程式碼中的註解：

```
iteration = 1
while True:

    # 先將當前的 Sim copy 一份儲存在 last_sim
    last_sim = []
    for list in sim:
        copy = list.copy()
        last_sim.append(copy)

    for i in range(vertex_size):
        for j in range(vertex_size):
            if i != j:

                sim_sum = 0

                if in_neighbors_num[i] != 0 and in_neighbors_num[j] != 0:
                # 若 i 有 parent，j 也有 parent

                    for k in in_neighbor[i]:
                    # i 的 parent k

                        for l in in_neighbor[j]:
                        # j 的 parent l

                            # 將上一輪計算得到的 Sim(k,l)加總起來
                            sim_sum += last_sim[k][l]

                    # 帶入公式更新 Sim(i,j)
                    sim[i][j] = (decay_factor / (in_neighbors_num[i] *
                                 in_neighbors_num[j])) * sim_sum

            else:
                # 對角線的值都是 1(∵ Sim(i,i)=1)
                sim[i][j] = 1

    # 依據 max iteration 決定是否 break
    if iteration >= max_iteration:
        break
    else:
        iteration += 1
```

PART 4 RESULT ANALYSIS AND DISCUSSION

以下 Graph 1-4 的結果皆為在參數 max_iteration 30 (每個演算法都總共執行 30 個 Iterations)、Decay Factor 0.7、Damping Factor 0.1 所執行的結果。

4.1 Graph 1

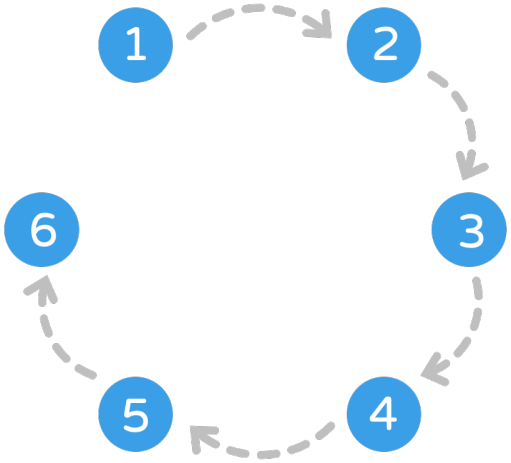

Figure 6. Graph 1 示意圖

Table 7. Graph 1 執行 HITS 和 PageRank 的結果

	Node 1	Node 2	Node 3	Node 4	Node 5	Node 6
Authority	0	0.2	0.2	0.2	0.2	0.2
Hub	0.2	0.2	0.2	0.2	0.2	0
PageRank	0.056	0.107	0.152	0.193	0.23	0.263

Table 8. Graph 1 執行 SimRank 的結果

	Node 1	Node 2	Node 3	Node 4	Node 5	Node 6
Node 1	1	0	0	0	0	0
Node 2	0	1	0	0	0	0
Node 3	0	0	1	0	0	0
Node 4	0	0	0	1	0	0
Nod3 5	0	0	0	0	1	0
Node 6	0	0	0	0	0	1

HITS：

- 由於 Authority 的計算方式是將 Parent 的 Hub 加總，由於 Node 1 沒有 Parent，所以 Node 1 的 Authority 就是 0。
- 同理，由於 Hub 的計算方式是將 Children 的 Authority 加總，由於 Node 6 沒有 Children，所以 Node 6 的 Hub 是 0。
- 除了 Node 1 和 6，其他 Node 的 In-Degree 和 Out-Degree 都是 1，所以其他 Node 的 Authority 和 Hub 都是 1/5 = 0.2。

PageRank：

- 由於 Graph 1 的 Edge 全部串在一起剛好是一個從 Node 1 連到 Node 6 的 Edge，所以 PageRank 的傳遞會從 Node 1 傳遞到 Node 6，最後累積在 Node 6。所以可以看到從 Node 1 到 Node 6，PageRank 呈現遞增的結果。

SimRank：

- 由於 Graph 1 的 Node 彼此間都沒有共同的 Parent，不同 Node 之間的 Similarity 不管怎麼計算都會是 0，所以最終的結果是個 Identity Matrix。

4.2 Graph 2

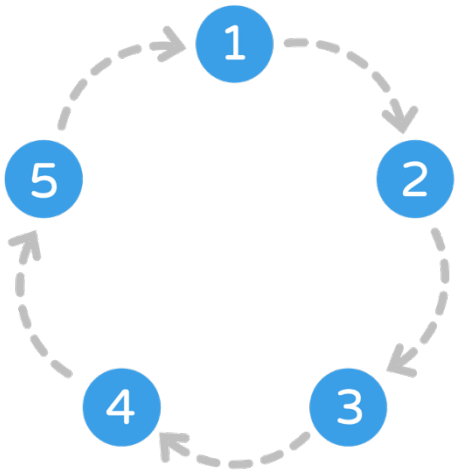

Figure 7. Graph 2 示意圖

Table 9. Graph 2 執行 HITS 和 PageRank 的結果

	Node 1	Node 2	Node 3	Node 4	Node 5
Authority	0.2	0.2	0.2	0.2	0.2
Hub	0.2	0.2	0.2	0.2	0.2
PageRank	0.2	0.2	0.2	0.2	0.2

Table 10. Graph 2 執行 SimRank 的結果

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

HITS：

- 由於 Graph 2 是一個環狀，且每個 Node 的 In-Degree 和 Out-Degree 都是 1，所以可以看到每個 Node 的 Authority 和 Hub 都是 1/Node 總數。

PageRank：

- 由於 Graph 2 是一個環狀，PageRank 不會像 Graph 1 最後都累積在某一個 Node，而是會平均分散在每個 Node，所以每個 Node 的 PageRank 也是 1/Node 總數。

SimRank：

- 由於 Graph 2 的這 5 個 Node，彼此都沒有共同的 Parent，所以不同的 Node 的 Similarity 都是 0，僅有 Sim(i, i)是 1。

4.3 Graph 3

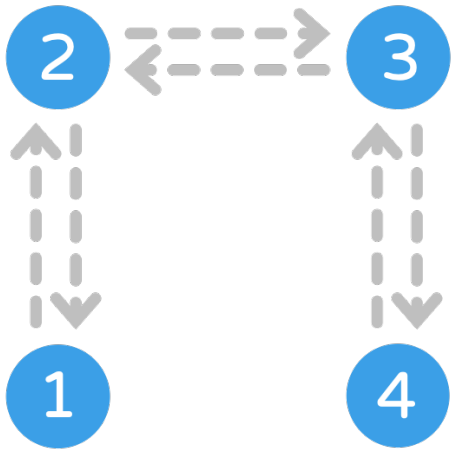

Figure 8. Graph 3 示意圖

Table 11. Graph 3 執行 HITS 和 PageRank 的結果

	Node 1	Node 2	Node 3	Node 4
Authority	0.190983	0.309017	0.309017	0.190983
Hub	0.190983	0.309017	0.309017	0.190983
PageRank	0.172414	0.327586	0.327586	0.172414

Table 12. Graph 3 執行 SimRank 的結果

	Node 1	Node 2	Node 3	Node 4
Node 1	1	0	0.538462	0
Node 2	0	1	0	0.538462
Node 3	0.538462	0	1	0
Node 4	0	0.538462	0	1

HITS：

- 可以看到由於 Node 2 和 3 的 In-Degree 和 Out-Degree 都是 2，Node 1 和 4 的 In-Degree 和 Out-Degree 都是 1，所以 Node 2 和 3 的 Hub 和 Authority 都相同，且都大於 Node 1 和 4。

PageRank：

- 由於 Node 2 和 3 的 In-Degree 都是 2，大於 Node 1 和 4 的 In-Degree，所以 PageRank 在傳遞的過程中，會逐漸累積在 Node 2 和 3，所以這兩個 Node 的 PageRank 大於 Node 1 和 4。

SimRank：

- 由於 Node 1 和 3 有共同的 Parent，Node 2，所以 $Sim(1,3)$ 和 $Sim(3,1)$ 出現了非 0 的結果，且 $Sim(1,3) = Sim(3,1)$ 。
- 同理，由於 Node 2 和 Node 4 有共同的 Parent，Node 3，所以 $Sim(2,4)$ 和 $Sim(4,2)$ 相等且非 0。
- 最後，由於 Node 2 和 3 的 In-Degree 和 Out-Degree 相等，且 Node 1 和 4 的 In-Degree 和 Out-Degree 相等，所以 $Sim(1,3) = Sim(3,1) = Sim(2,4) = Sim(4,2)$。

4.4 Graph 4

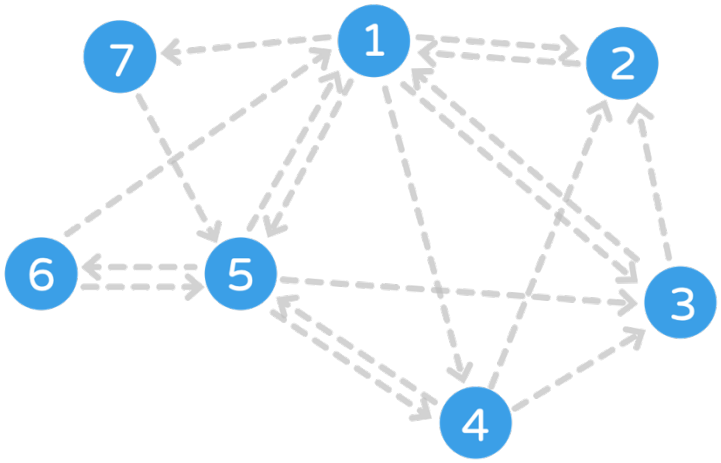

Figure 9. Graph 4 示意圖

Table 13. Graph 4 執行 HITS 和 PageRank 的結果

	Node 1	Node 2	Node 3	Node 4	Node 5	Node 6	Node 7
Authority	0.139	0.178	0.201	0.140	0.201	0.056	0.084
Hub	0.275	0.048	0.109	0.199	0.184	0.117	0.069
PageRank	0.288	0.161	0.139	0.107	0.183	0.055	0.066

Table 14. Graph 4 執行 SimRank 的結果

	Node 1	Node 2	Node 3	Node 4	Node 5	Node 6	Node 7
Node 1	1	0.243	0.232	0.239	0.221	0.303	0.175
Node 2	0.243	1	0.294	0.256	0.295	0.17	0.343
Node 3	0.232	0.294	1	0.34	0.275	0.339	0.341
Node 4	0.239	0.256	0.34	1	0.23	0.427	0.427
Node 5	0.221	0.295	0.275	0.23	1	0.159	0.3
Node 6	0.303	0.17	0.339	0.427	0.159	1	0.155
Node 7	0.175	0.343	0.341	0.427	0.3	0.155	1

Table 15. 每個 Node 的 Parent、Children、以及排序資訊

Node	Parent	Children	Authority Ranking	Hub Ranking	PageRank Ranking
1	2, 3, 5, 6	2, 3, 4, 5, 7	5	1	1
2	1, 3, 4	1	3	7	3
3	1, 4, 5	1, 2	2	5	4
4	1, 5	2, 3, 5	4	2	5
5	1, 4, 6, 7	1, 3, 4, 6	1	3	2
6	5	1, 5	7	4	7
7	1	5	6	6	6

HITS

- 可以看到 Node 1 雖然 Parent 數量很高，但推薦 Node 1 的是 Node 2、3、5、6，但這幾個都不算是很好的推薦者(Hub 都不高)，所以 Authority 排名沒有很好；但因為 Node 1 幾乎推薦了每一個 Node，所以他的 Hub 排名很高。
- 從 Figure 10 也可以看到，Node 1 和 5 在第一個 Iteration 因為 Parent 數量高，所以 Authority 最高，但在第二個 Iteration，Node 1 的 Authority 馬上就掉下來。
- Node 5 則因為推薦他的 Node 的 Hub 排名都滿好的，有 Hub 排名的第一、第二、和第四名，所以他的 Authority 才會排很前面。
- Node 2 因為只有推薦 Node 1，但 Node 1 的 Authority 排名第五，沒有很好，所以

才導致 Node 2 變成 Hub 最後一名。

- 從 Figure 10 和 Figure 11 可以看到，Authority 和 Hub 在每一個 Iteration 都是上下震盪，之後才趨於平穩。會有這現象是因為這一回合的 Authority 是由上一個 Iteration 的 Parent 的 Hub 而來；而這一回合的 Hub 是由上一個 Iteration 的 Children 的 Authority 而來。但這一回合的 Authority/Hub 又會影響到下一回合的 Children/Parent 的 Hub/Authority。所以其實是有兩條路徑在影響這些數字，所以才會看到每一個 Iteration 會這樣上上下下的現象。

PageRank

- 可以看到 PageRank 的排名跟每個 Node 的 Parent 數量有一定相關性，因為 Parent 越多，就代表有越多的 PageRank 會傳遞到這個 Node。

SimRank

- 相似度最高的是 $Sim(4,6)$ 和 $Sim(4,7)$，0.427，他們共同的 Parent 分別是 1 或 5。
- 第二高的則是 $Sim(3,6)$，0.339，其共同 Parent 是 5。
- 第三高的則是 $Sim(2,7)$，0.343，其共同 Parent 是 1。
- 第四高的則是 $Sim(3,7)$，0.341，其共同 Parent 是 1。
- 從這幾點可以觀察到，若某個 Node 的 Parent，是另一個 Node 的 Parent 的子集，那他們的 Similarity 就會很高。此外，若某兩個 Node 的 Parent 的差集數量越小，那他們的 Similarity 也會越大。
- 若某兩個 Node 的 Parent 的交集是空集合，那他們的 Similarity 就會很低，譬如說 $Sim(6,7) = 0.155$ 和 $Sim(6,2) = 0.17$。雖然 Node 6 和 Node 2 的 Parent 都沒有交集，但 Node 2 的 Parent 的 Parent 就有 Node 5，也就是 Node 6 的 Parent，所以 $Sim(6,2)$ 大於 $Sim(6,7)$。

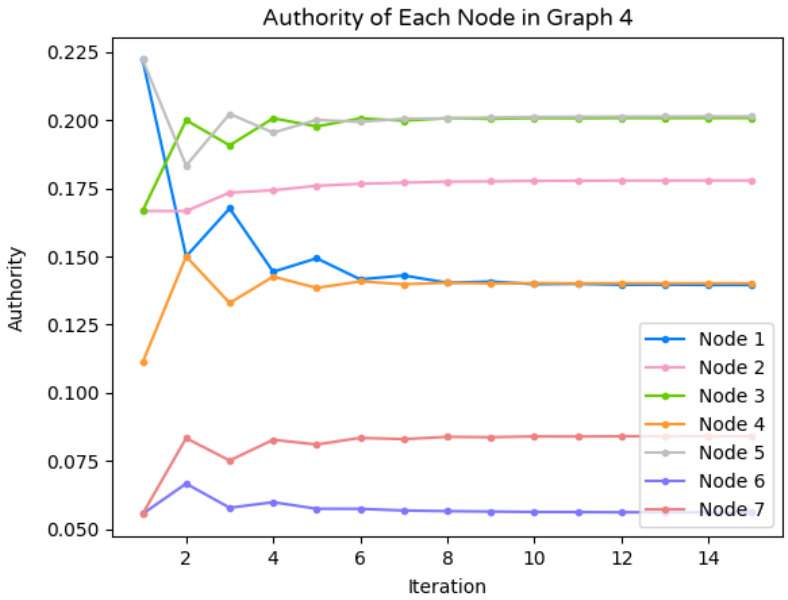

Figure 10. Graph 4 每個 Node 的 Authority 變化

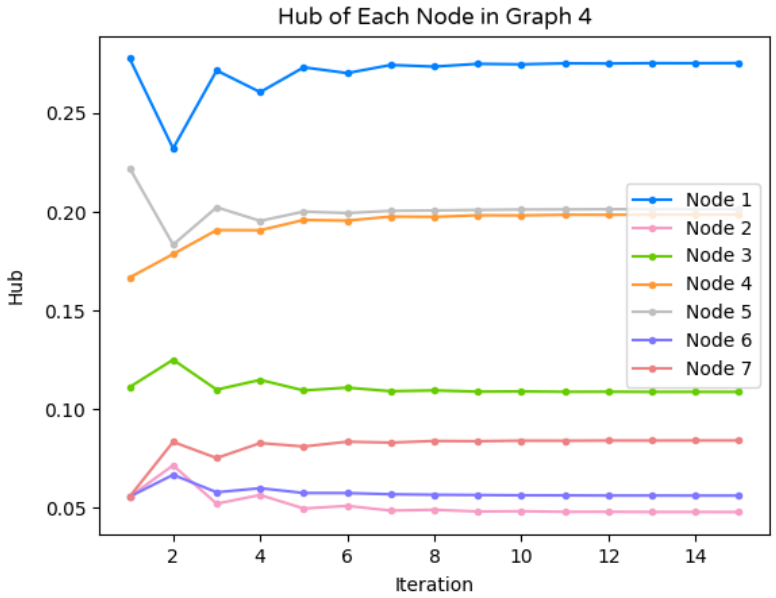

Figure 11. Graph 4 每個 Node 的 Hub 變化

4.5 Damping Factor

為了要解決 Rank Sink Problem，也就是假如某個 Node 沒有 Parent 而他永遠就都沒有 Rank；或者是某個 Node 沒有 Children，導致該 Node 無法將 Rank 傳給其他人，因此用 Damping Factor 來解決這問題。譬如說 Damping Factor 0.1，就代表有 0.1 的機會是隨機跳到該 Node，0.9 的機會是由該 Node 的 Parent 跳到該 Node。

實驗一

測試資料：

由於 Graph 1 的 Node 1 沒有 Parent，Node 7 沒有 Children，因此以 Graph 1 做測試。

實驗條件&步驟：

- 測試有 Damping Factor 0.1 以及無 Damping Factor，參數 max_iteration 1000000，收斂門檻 epsilon 1e-10，並以 `numpy.allclose(前一輪 page_rank, 目前 page_rank, atol = epsilon)`來確認是否達到收斂。
- 對 Graph 1 執行 PageRank，並記錄達到收斂的 Iteration。

結果與討論：

Table 16. 對 Graph 1 執行有或無 Damping Factor 的 PageRank

	Node 1	Node 2	Node 3	Node 4	Node 5	Node 6	Converge Iteration
Damping 0.1	0.056	0.107	0.152	0.193	0.23	0.263	7
No Damping	0	0	0	0	0	0	7

– 從 Table 16 可以看到，沒有使用 Damping Factor 的話，由於 Graph 1 的 Node 1 無 Parent，Node 6 無 Children，導致全部的 Node 的 PageRank 都是 0。
這是因為 Node 1 沒有 Parent，所以 Node 1 不可能得到 PageRank，再加上 Graph 1 的 Edge 全部是一整條單一方向，所以其他的 Node 若要得到 PageRank 只能仰賴 Node 1 傳送的 PageRank，但因 Node 1 沒有 Parent，所以全部 Node 的 PageRank 都是 0。

– 使用 Damping Factor 0.1 之後，每個 Node 都有 0.1 的機會是隨機進入該 Node 的，所以 Node 1 的機率不再是 0。
另外，由於 Node 6 沒有 Children，導致 PageRank 在傳送的過程，大部分都累積到 Node 6，所以可以看到 PageRank 從 Node 1 到 Node 6 呈現遞增的現象。

實驗二

目的：

欲觀察若無 Damping Factor，則對於含有無 Children 的 Node 的 Graph 執行 PageRank 會有什麼結果。

測試資料：

將 Graph 1 做修改，如 Figure 12 所示(橘色的 Edge 是新加入的 Edge)。

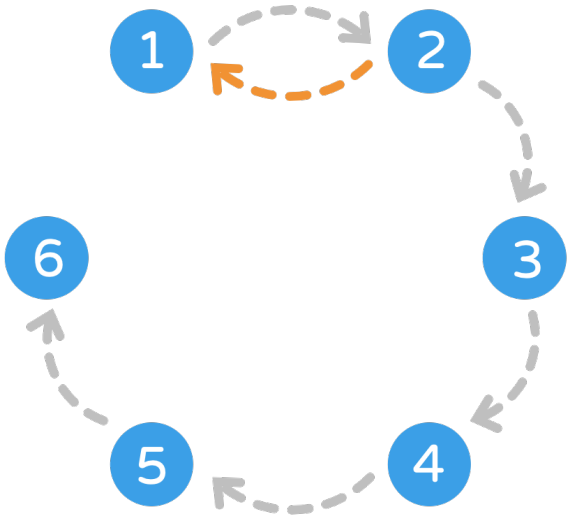

Figure 12. 經修改後含有無 Children 的 Node 的 Graph 1

實驗條件&步驟：

- 測試有 Damping Factor 0.1 以及無 Damping Factor，參數 max_iteration 1000000，收斂門檻 epsilon 1e-10，並以 `numpy.allclose(前一輪 page_rank, 目前 page_rank, atol = epsilon)` 來確認是否達到收斂。
- 對測試資料執行 PageRank，並記錄達到收斂的 Iteration。

結果與討論：

Table 17. 含有無 Children 的 Node 執行 PageRank 的結果

	Node 1	Node 2	Node 3	Node 4	Node 5	Node 6	Converge Iteration
Damping 0.1	0.124	0.163	0.124	0.163	0.197	0.229	29
No Damping Factor							
Epsilon 1e-10	0.083	0.167	0.083	0.167	0.167	0.333	63
Epsilon 1e-20	0.083	0.167	0.083	0.167	0.167	0.333	129
Epsilon 1e-50	0.083	0.167	0.083	0.167	0.167	0.333	329
Epsilon 1e-100	0.083	0.167	0.083	0.167	0.167	0.333	661

- 在有 Damping factor 的情況下，可以看到 PageRank 都是從 Node 1 到 Node 6 呈現遞增的狀況，PageRank 會累積在編號比較後面的 Node。
- 但若未使用 Damping Factor，且 Graph 中存在無 Children 的 Node 則會得到比較特別的狀況。從 Table 17 可以看到，只要將收斂的門檻 epsilon 不停調低，達到收斂的 Iteration 就會不停增加，且不管是設定哪個收斂的門檻，PageRank 的結果都一樣。這在前面的實驗、或者是有使用 Damping Factor 的狀況都沒有發生過。
- 這狀況是因為若存在無 Children 的 Node，則 Adjacency Matrix 的某一列就都會是 0，這樣就會無解，所以才會看到這個情況。

實驗三

目的：

欲觀察若無 Damping Factor，則對存在無 Parent 的 Node 的 Graph 執行 PageRank 會有什麼結果。

測試資料：

將 Graph 1 做修改，如 Figure 12 所示(橘色的 Edge 是新加入的 Edge)。

實驗條件&步驟：

- 測試有 Damping Factor 0.1 以及無 Damping Factor，參數 max_iteration 1000000，收斂門檻 epsilon 1e-10，並以 `numpy.allclose(前一輪 page_rank, 目前 page_rank, atol = epsilon)` 來確認是否達到收斂。
- 對測試資料執行 PageRank，並記錄達到收斂的 Iteration。

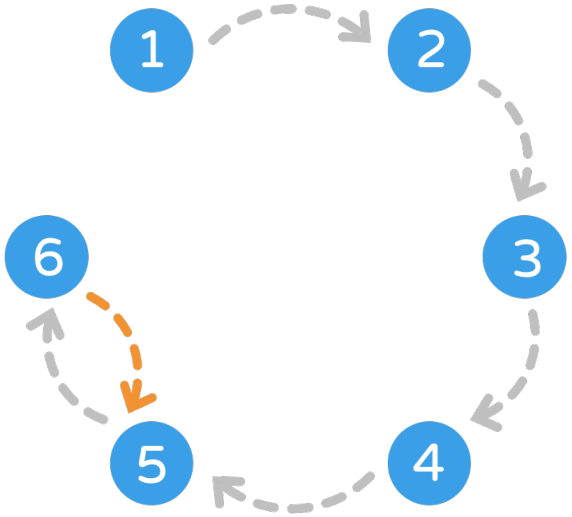

Figure 13. 經修改後含有無 Parent 的 Node 的 Graph 1

結果與討論：

Table 18. 含有無 Parent 的 Node 執行 PageRank 的結果

	Node 1	Node 2	Node 3	Node 4	Node 5	Node 6	Converge Iteration
Damping 0.1	0.017	0.032	0.045	0.057	0.438	0.411	5
No Damping	0	0	0	0	0.5	0.5	5

- 可以看到在有 Damping Factor 的狀況下，PageRank 都是從 Node 1 到 Node 6 呈現遞增的狀況，PageRank 會累積在編號比較後面的 Node。
- 在沒有使用 Damping Factor 的狀況下，可以看到 PageRank 集中在最後的兩個 Node。
- 若將無 Damping Factor 在每一個 Iteration 的 PageRank 的變化拿出來看，則可看到如 Table 19 的變化。

  可以看到無 Parent 的 Node，即 Node 1，在第一個 Iteration 其 PageRank 就是 0。所以對於無 Parent 的 Node，若沒有使用 Damping Factor，其 PageRank 就會是 0。

  再來，由於 Figure 13 的特性，PageRank 會從 Node 2 傳到 Node 3、再傳到 Node 4、最後傳到 Node 5 和 Node 6，因為無法再傳出去，所以就在 Node 5 和 6 之間傳遞。

Table 19. 無 Damping Factor 每一輪的執行結果

	Node 1	Node 2	Node 3	Node 4	Node 5	Node 6
Iteration 1	0	0.167	0.167	0.167	0.333	0.167
Iteration 2	0	0	0.167	0.167	0.333	0.333
Iteration 3	0	0	0	0.167	0.5	0.333
Iteration 4	0	0	0	0	0.5	0.5
Iteration 5	0	0	0	0	0.5	0.5

實驗四

目的：

欲測試不同的 Damping Factor，對於多少 Iteration 才會達到收斂是否會有影響。

測試資料：

Graph 1-6，以及 ibm-5000。

實驗條件&步驟：

- 測試不同 Damping Factor 0.1、0.2、0.3、0.4、0.5、0.6、0.7、0.8、0.9。

- 參數 max_iteration 1000000，收斂門檻 epsilon 1e-10，並以 `numpy.allclose(前一輪 page_rank, 目前 page_rank, atol = epsilon)`來確認是否達到收斂。
- 對測試資料執行 PageRank，並記錄達到收斂的 Iteration。

結果與討論：

- 從 Figure 14 可以看到，基本上隨著 Damping Factor 增加，達到收斂的 Iteration 就會越少。
- 比較特別的是 Graph 2 的結果，不管用哪個 Damping Factor，都是在第一個 Iteration 就達到收斂，這是因為 Graph 2 是一個環狀，所以 PageRank 不管怎麼傳，每一輪的 PageRank 都會跟前一輪的一樣，所以在第一輪就會達到收斂。
- 從 PageRank 的公式可以知道，當 Damping Factor 越大，就代表「隨機」達到該 Node 的機率就越大、從 Parent 而來的機率就越小、Parent 的影響力也越小，所以會更快達到收斂。

另外，Table 20 和 Table 21 是 Graph 3 和 Graph 4 分別在 Damping Factor 0.1 和 0.9 達到收斂時的 PageRank。

- 可以看到，當 Damping Factor 越大，每個 Node 的 PageRank 就越平均，差異越小，這也是因為隨機到達每個 Node 的機會比較大，從 Parent 而來的機會比較小，所以 Parent 傳遞的 PageRank 的影響力就越小，反而「隨機」這個因素比 Parent 還有影響力，所以最終的 PageRank 會更趨近於 1/(Node 總數)。

Table 20. Graph 3 在 Damping Factor 0.1 和 0.9 達到收斂時的 PageRank

Graph 3	Node 1	Node 2	Node 3	Node 4
Damping 0.1	0.172	0.328	0.328	0.172
Damping 0.9	0.238	0.262	0.262	0.238

Table 21. Graph 4 在 Damping Factor 0.1 和 0.9 達到收斂時的 PageRank

Graph 4	Node 1	Node 2	Node 3	Node 4	Node 5	Node 6	Node 7
Damping 0.1	0.288	0.161	0.139	0.107	0.183	0.055	0.066
Damping 0.9	0.16	0.143	0.14	0.136	0.156	0.132	0.132

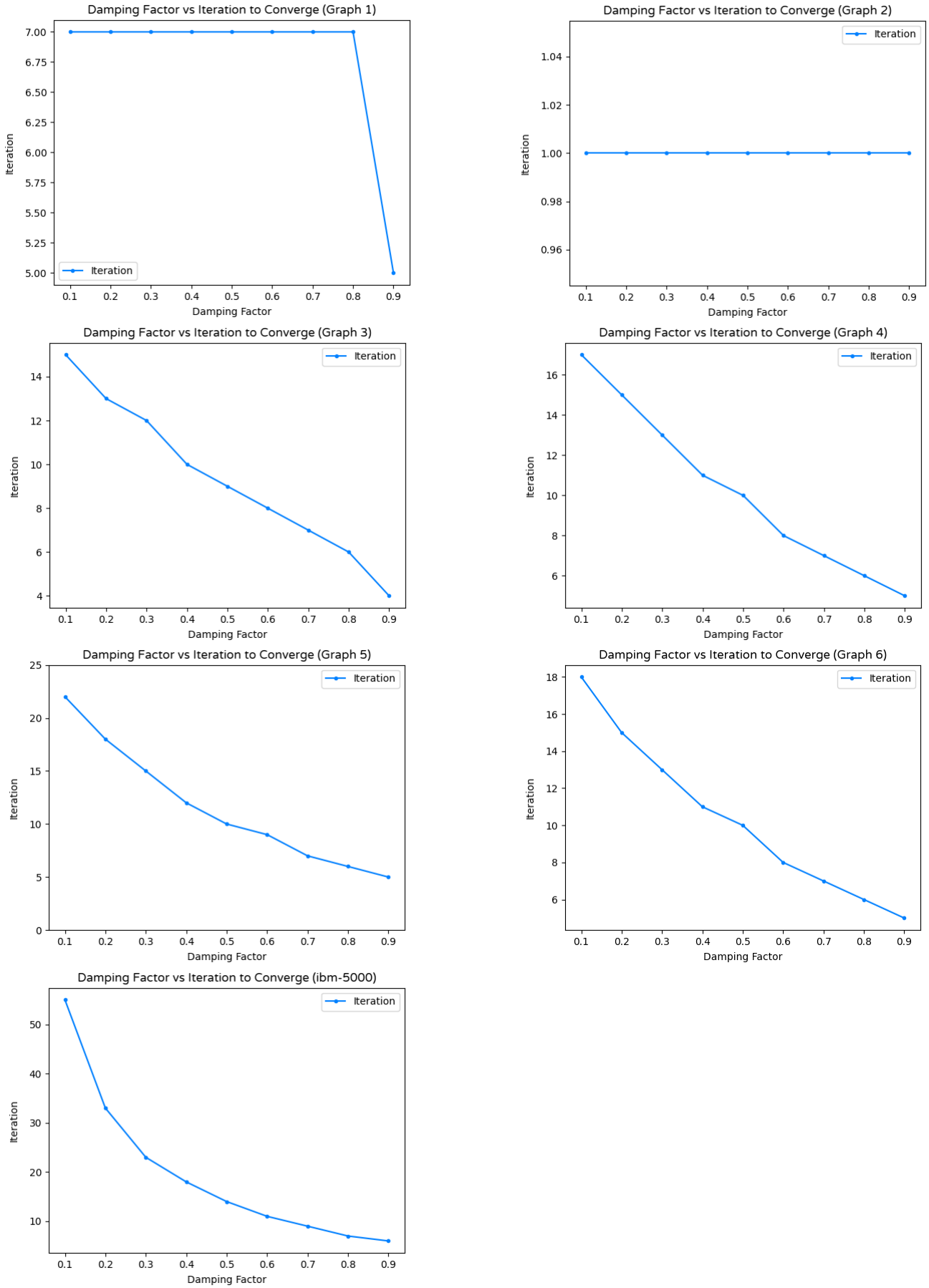

Figure 14. 不同 Damping Factor 達到收斂的 Iteration

4.6 Decay Factor

實驗五

目的：

由 SimRank 的公式可以看到，若 Decay Factor 越小，Similarity 就越小，所以我認為若 Decay Factor 越小，就會越快達到收斂，因為前一輪跟後一輪的差距會比較快變小。為了證實這個想法，執行以下實驗。

測試資料：

Graph 1-6，以及 ibm-5000。

實驗條件&步驟：

- 對測試資料執行 SimRank，Decay Factor 0.1、0.2、0.3、0.4、0.5、0.6、0.7、0.8、0.9。
- max_iteration 100000，收斂門檻 epsilon 1e-10，並以 `numpy.allclose(前一輪 sim_rank, 目前 sim_rank, atol = epsilon)`來確認是否達到收斂，並紀錄不同 Decay Factor 達到收斂的 Iteration，結果如 Figure 15 所示。

結果與討論：

- 可以看到 Graph 3-6，達到收斂的 Iteration 都會隨著 Decay Factor 的增加而增加，跟原本的猜測一樣。
- 但在 Graph 1, 2、以及 ibm-5000，卻是分別在第 1 個 Iteration 和第 4 個 Iteration 就達到收斂，且不隨著 Decay Factor 的增加而有所變動。
  我覺得是因為這幾個 Graph 的 SimRank 都是稀疏矩陣，尤其像 Graph 1 和 2 的 SimRank 是 Identity Matrix，而 ibm-5000 的 SimRank 除了對角線，大部分也都是 0，僅有少部分非 0。
  所以可以看到，對於相似度越低的 Graph (Node 幾乎都沒有共同 Parent)，Decay Factor 對這種 Graph 的影響不大、也不太影響他們達到收斂的 Iteration。

接下來，我想看不同 Decay Factor 對於達到收斂的 SimRank 會有什麼影響，因此我將前述實驗 Graph 3 在 Decay Factor 0.1 和 0.9 最終收斂的 SimRank 列出來，如 Table 22 和 Table 23 所示。

Table 22. Graph 3 在 Decay Factor 0.1 達到收斂時的 SimRank

	Node 1	Node 2	Node 3	Node 4
Node 1	1	0	0.052632	0
Node 2	0	1	0	0.052632
Node 3	0.052632	0	1	0
Node 4	0	0.052632	0	1

Table 23. Graph 3 在 Decay Factor 0.9 達到收斂時的 SimRank

	Node 1	Node 2	Node 3	Node 4
Node 1	1	0	0.818177	0
Node 2	0	1	0	0.818177
Node 3	0.818177	0	1	0
Node 4	0	0.818177	0	1

- 從 Table 22 和 Table 23 可以看到，達到收斂的 SimRank 的數字會隨著 Decay Factor 的增加而增加。這個結果也並不意外，因為從公式就可以看到 Decay Factor 的大小也會影響到 Similarity 的大小。
- 至於 Graph 1 和 2 的 SimRank，則是不論 Decay Factor 為何，都是 Identity Matrix。

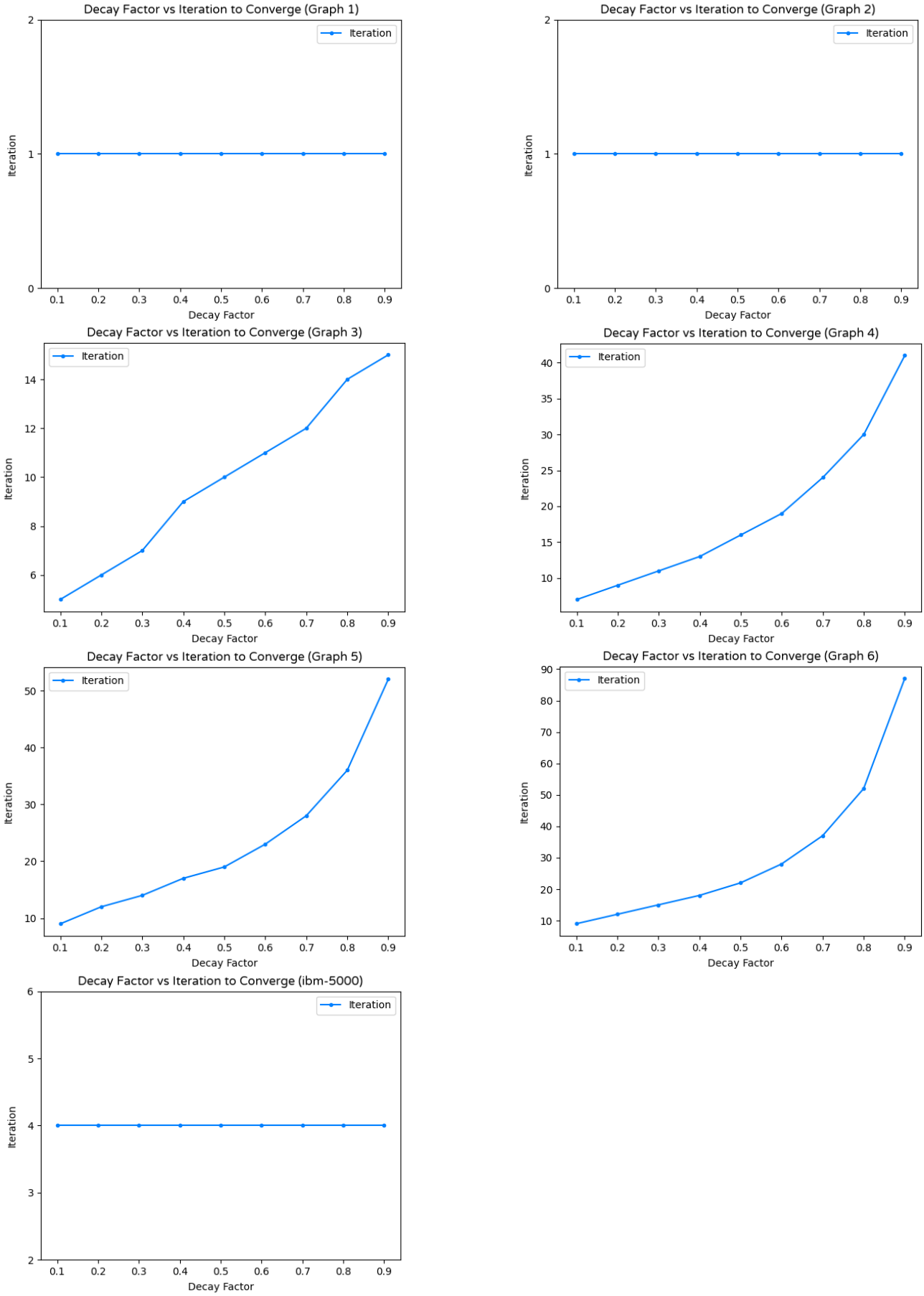

Figure 15. 不同 Decay factor 對於達到收斂 iteration 的影響

PART 5 EFFECTIVENESS ANALYSIS

5.1 Execution Time of the 3 Algorithms

測試資料：

Graph 1-6, 以及 ibm-5000

由於這三個演算法的執行時間會跟測試資料的 Node 數量以及 Edge 數量有關, 因此 Table 24 列出這 7 個測試資料的 Edge 和 Node 數量。

Table 24. 測試 Data 的 Node 與 Edge 數量

	Node Number	Edge Number
Graph 1	6	5
Graph 2	5	5
Graph 3	4	6
Graph 4	7	18
Graph 5	469	1102
Graph 6	1228	5220
ibm-5000	836	4798

實驗條件&步驟：

對這 7 個資料執行 HITS、PageRank、SimRank 演算法, 參數 max_iteration 30 (三個演算法都各跑 30 Iteration)、Damping Factor 0.1、Decay Factor 0.7, 結果如 Table 25、Figure 16、以及 Figure 17 所示。

Table 25. 三個演算法的執行時間(單位：秒)

	HITS	PageRank	SimRank
Graph 1	0.00153	0.00015	0.00064
Graph 2	0.00122	0.00012	0.00058
Graph 3	0.00121	0.00011	0.00057
Graph 4	0.00234	0.00022	0.00224
Graph 5	1.70784	0.06332	9.5617
Graph 6	11.7193	0.39496	141.19387
ibm-5000	5.72715	0.18675	75.91552

結果與討論：

- 可以看到, 這三個演算法基本上都隨著 Node 或 Edge 數量的增加, 執行時間也會因此增加。

- 三個演算法當中，以 PageRank 所需的時間最少，因為 PageRank 的程式碼，處理的資料都儲存在一維矩陣。
  此外，執行 PageRank 會用到的一些資訊我都有事先記錄下來，而並非在每個 for loop 裡面才去尋找，像是每個 Node 有哪些 Parent，以及每個 Parent 有多少 Children 等資訊。
- 從 Figure 16 可以看到，當 Node 數量少時，HITS 所需的執行時間比 SimRank 還多；但在 Figure 17 可以看到，當 Node 數量變多時，SimRank 所需的時間就會急劇增加。
  本來以為我 SimRank 用了 4 層 for loop 去執行，應該會花最多時間。測試 HITS 程式碼後，發現是在計算 Authority 和 Hub 的地方花最多時間，這個地方我是使用 numpy.dot()來計算。
  所以可以看到當 Node 和 Edge 數量少的時候，numpy.dot()所需的時間會比 SimRank 的 4 層 for loop 還多；但當 Node 和 Edge 數量增加時，SimRank 的 4 層 for loop 就會造成時間急劇增加。

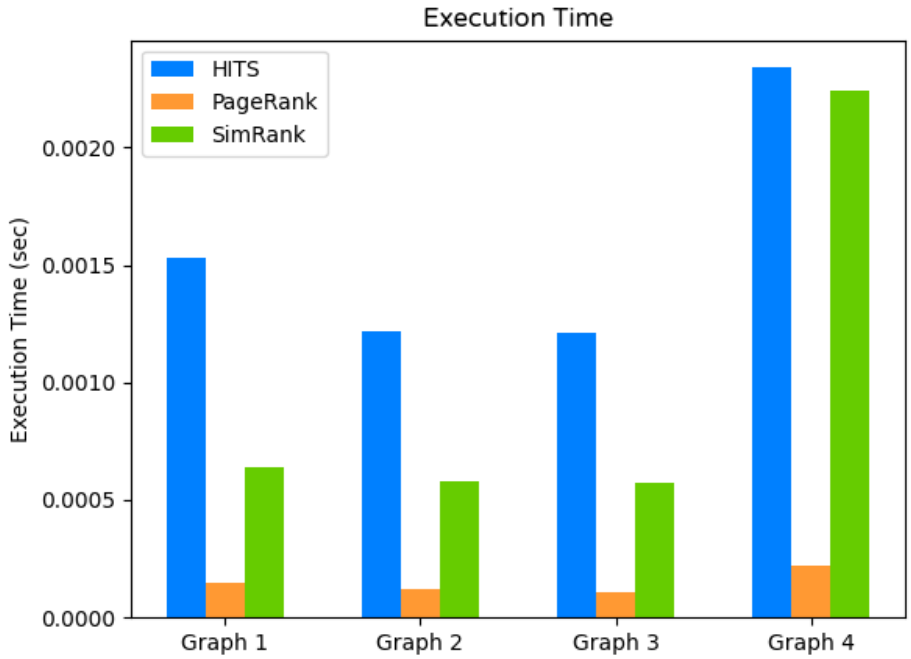

Figure 16. Graph 1-4 執行三個演算法的時間

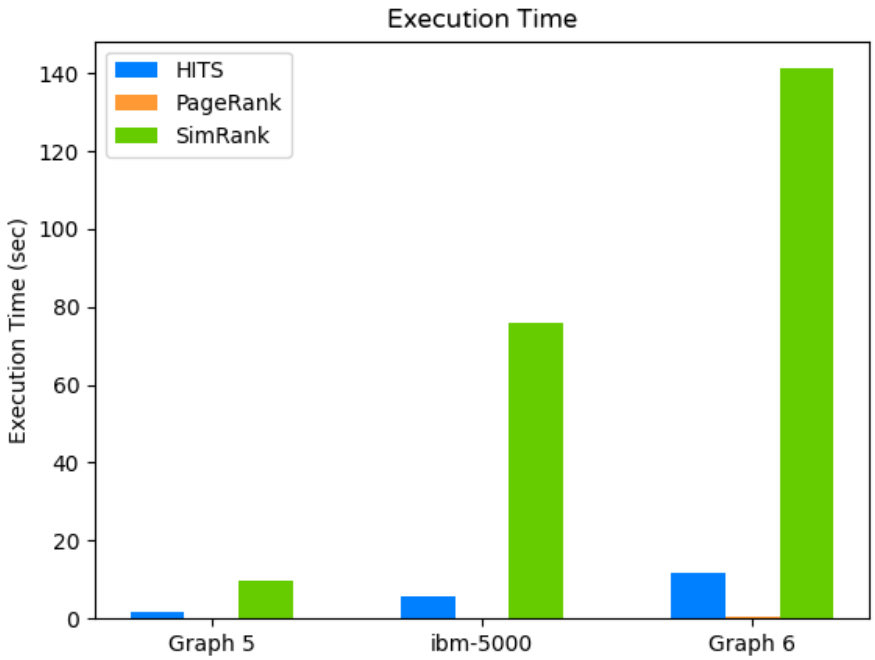

Figure 17. Graph 5、6、以及 ibm-5000 執行三個演算法的時間

實驗六

目的：

為了測試若不用 numpy.dot()計算 Authority 和 Hub 是否能改善執行時間，我修改我的 HITS 程式碼，並測試新的 HITS 程式碼的執行時間。

測試資料：

Graph 1-6、以及 ibm-5000。

實驗條件&步驟：

對 7 個測試資料執行新的 HITS 程式碼，每個測試資料都跑 30 個 Iteration，並記錄執行時間。

新的 HITS 程式碼說明如下：

首先，將每個 Vertex 的 Authority 和 Hub 都初始化成 1：

```
# 初始化 hub 和 authority
hub = [1 for x in range(vertex_size)]
authority = [1 for x in range(vertex_size)]
```

接下來則是紀錄每個 Node 的 Parent 和 Children，以方便後面使用：

```
# 先將每個 node 的 parent/children 的資訊記錄下來
parent_list = []
```

```
children_list = []
for i in range(vertex_size):
    parent = []
    children = []

    for j in range(vertex_size):
        if adj_matrix[j][i] == 1:
            parent.append(j)
        if adj_matrix[i][j] == 1:
            children.append(j)

    if len(parent) == 0:
        parent_list.append([-1])
    else:
        parent_list.append(parent)

    if len(children) == 0:
        children_list.append([-1])
    else:
        children_list.append(children)
```

再來則是進入 while 迴圈，不停更新每個 Node 的 Authority 和 Hub，直到達到設定的終止條件，詳細說明如程式碼中的註解：

```
iteration = 1
while True:
    # 先 copy 一份當前的 hub 和 authority
    prev_authority = authority.copy()
    prev_hub = hub.copy()

    for i in range(vertex_size):
        # 計算 authority
        if parent_list[i][0] != -1:
            # 若 node i 有 parent
            new_auth = 0
            for j in parent_list[i]:
                # 把 node i 的每個 parent j 前一輪的 hub 值加總起來
                new_auth += prev_hub[j]
            # 更新 authority
            authority[i] = new_auth
        else:
            # 若 node i 沒有 parent，就直接 0
            authority[i] = 0

        # 計算 hub
        if children_list[i][0] != -1:
            # 若 node i 有 parent
            new_hub = 0
            for j in children_list[i]:
                # 把 node i 的每個 parent j 前一輪的 authority 值加總起來
                new_hub += prev_authority[j]
            # 更新 hub
```

```
            hub[i] = new_hub
        else:
            # 若 node i 沒有 parent，就直接 0
            hub[i] = 0

    # 記得要做 Normalization，這樣才會收斂
    authority = one_norm(authority, vertex_size)
    hub = one_norm(hub, vertex_size)

    # 依據 max iteration 決定是否 break
    if iteration >= max_iteration:
        break
    else:
        iteration += 1
```

結果與討論：

Table 26. 新的 HITS 演算法執行時間(單位：秒)

	HITS (Old)	HITS (New)	PageRank	SimRank
Graph 1	0.00153	0.00029	0.00015	0.00064
Graph 2	0.00122	0.00027	0.00012	0.00058
Graph 3	0.00121	0.00023	0.00011	0.00057
Graph 4	0.00234	0.0004	0.00022	0.00224
Graph 5	1.70784	0.08094	0.06332	9.5617
Graph 6	11.7193	0.98606	0.39496	141.19387
ibm-5000	5.72715	0.19853	0.18675	75.91552

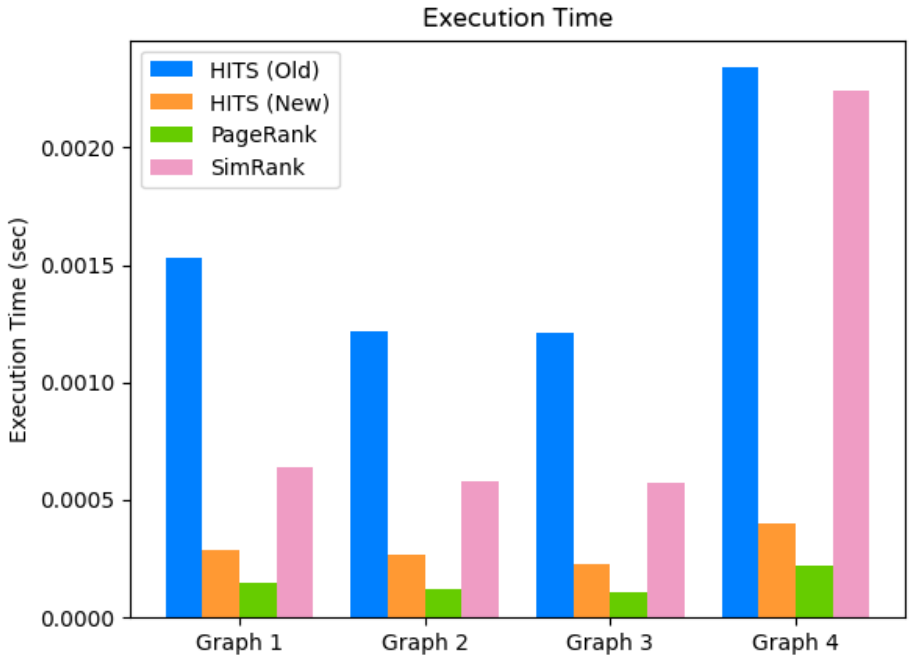

Figure 18. 新的 HITS 程式碼執行 Graph 1-4 的執行時間

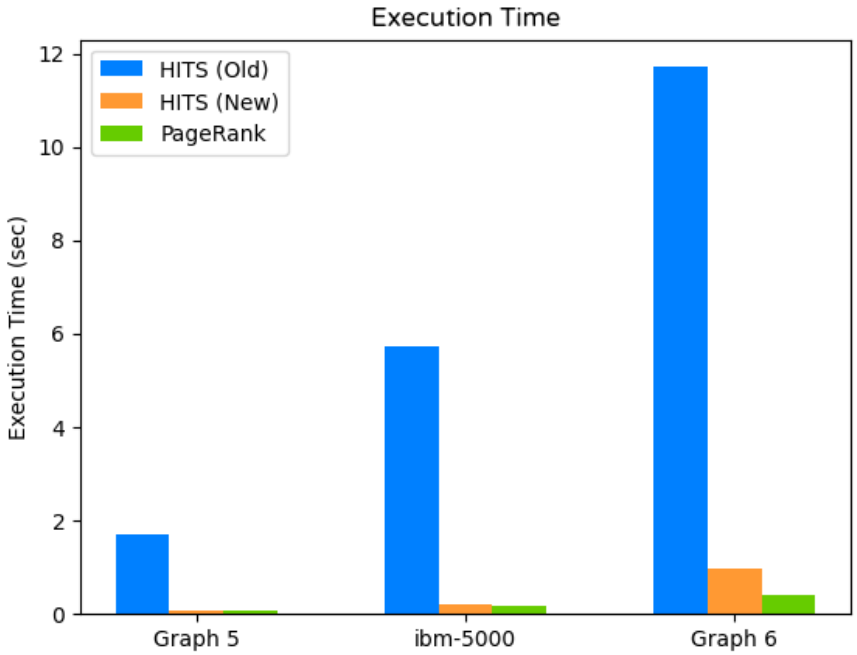

Figure 19. 新的 HITS 程式碼執行 Graph 5-6 和 ibm-5000 的執行時間

- 從 Figure 18 和 Table 26 可以看到，新的 HITS 所需的執行時間大幅減少許多，所需時間和 PageRank 差不多，只比 PageRank 多一些。
- 從這個例子可以看到，雖然使用 Package 的功能會方便許多，但有可能會因此增加一些執行時間。若我其他兩個演算法也都有用到其他 Package 的功能，而那些功能又會造成執行時間增加，那有可能就會誤判這三個演算法的效能。

5.2 The Effect of Edge Number on Execution Time

實驗七

目的：

欲測試不同 Edge 數量對執行時間的影響。

測試資料：

用以下 5 個指令生出 5 份 ibm 資料。

```
./gen lit  -ntrans 0.5  -tlen 10  -nitems 0.002  -npats 5  -conf 0.2  -patlen 20
./gen lit  -ntrans 0.5  -tlen 10  -nitems 0.004  -npats 5  -conf 0.2  -patlen 20
./gen lit  -ntrans 0.5  -tlen 10  -nitems 0.01   -npats 5  -conf 0.2  -patlen 20
./gen lit  -ntrans 0.5  -tlen 10  -nitems 0.02   -npats 5  -conf 0.2  -patlen 20
./gen lit  -ntrans 0.5  -tlen 10  -nitems 0.1    -npats 5  -conf 0.2  -patlen 20
```

Table 27. 不同 nitems 參數產生的 Node 與 Edge 數量

	Node 數量	Edge 數量
nitems 0.002	354	692
nitems 0.004	354	1382
nitems 0.01	346	2417
nitems 0.02	345	3137
nitems 0.1	349	4333

實驗條件&步驟：

對這 7 個資料執行 HITS、PageRank、SimRank 演算法，參數 max_iteration 30（三個演算法都各跑 30 Iteration）、Damping Factor 0.1、Decay Factor 0.7，結果如 Figure 20 所示。

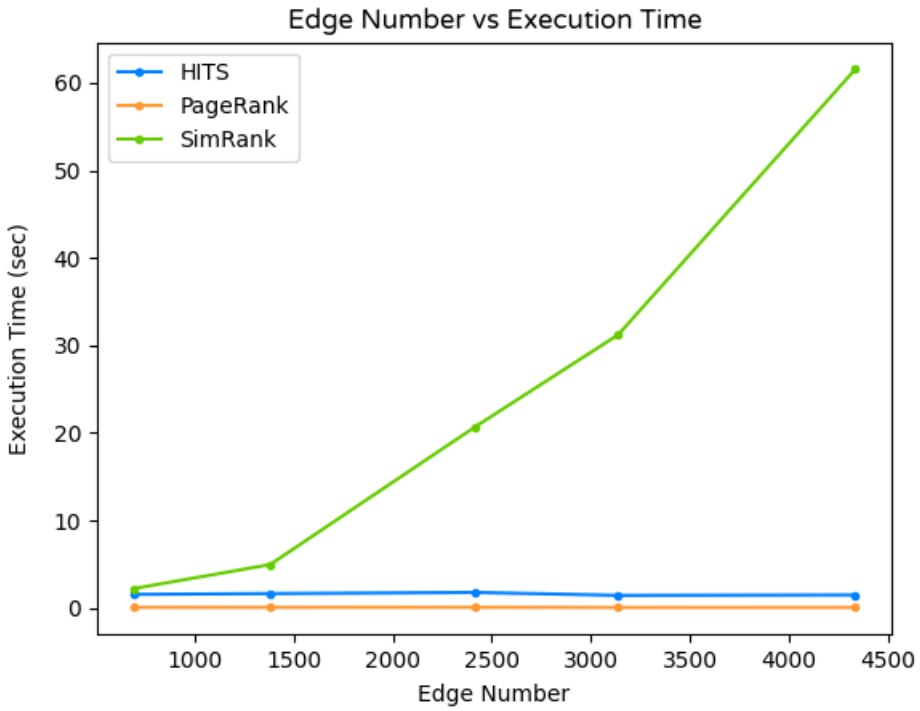

Figure 20. 不同 Edge 數量對演算法執行時間的影響

結果與討論：

- 可以看到 HITS 和 PageRank 不太會隨著 Edge 數量的增加而增加執行時間，因為他們要處理的資料都是儲存在一維陣列，而一維陣列的大小取決於 Node 數量，所以這兩個演算法的執行時間主要受 Node 數量影響。
- SimRank 則會隨著 Edge 數量增加而大幅增加執行時間，因為我用了 4 層 for loop 去執行，所以 SimRank 的執行時間對 Edge 數量很敏感。

5.3 The Effect of Node Number on Execution Time

實驗八

目的：

為了驗證我前面的猜測，想測試不同的 Node 數量對 HITS 和 PageRank 執行時間的影響。

測試資料：

用以下 5 個指令生出 5 份 ibm 資料。

```
./gen lit  -ntrans 0.5  -tlen 10  -nitems 0.01  -npats 5  -conf 0.2  -patlen 20
./gen lit  -ntrans 1    -tlen 10  -nitems 0.01  -npats 5  -conf 0.2  -patlen 20
./gen lit  -ntrans 1.5  -tlen 10  -nitems 0.01  -npats 5  -conf 0.2  -patlen 20
./gen lit  -ntrans 2    -tlen 10  -nitems 0.01  -npats 5  -conf 0.2  -patlen 20
./gen lit  -ntrans 2.5  -tlen 10  -nitems 0.01  -npats 5  -conf 0.2  -patlen 20
```

Table 28. 不同 ntrans 參數產生的 Node 和 Edge 數量

	Node 數量	Edge 數量	Edge/Node
ntrans 0.5	346	2417	6.99
ntrans 1	694	4870	7.02
ntrans 1.5	1005	7052	7.02
ntrans 2	1352	9499	7.03
ntrans 2.5	1703	11998	7.05

由於增加 Node 數量，Edge 數量勢必也會增加。為了減少 Edge 數量增加所造成的變異，僅能盡量控制 Node 和 Edge 維持在固定的比例，由 Table 28 可見 Edge 對 Node 的比例幾乎都是在 7。

實驗條件&步驟：

- 對這五筆測試資料執行 HITS、PageRank、以及 SimRank 演算法。
- 參數 Damping Factor 0.1、Decay Factor 0.7、max_iteration 30 (三個演算法都各跑 30 Iteration)。

結果與討論：

- 從 Figure 21 可以看到，三個演算法都會隨著 Node 數量增加而增加執行時間。
- 其中以 SimRank 所增加的時間最多，其次是 HITS，上升幅度最小的則是 PageRank。

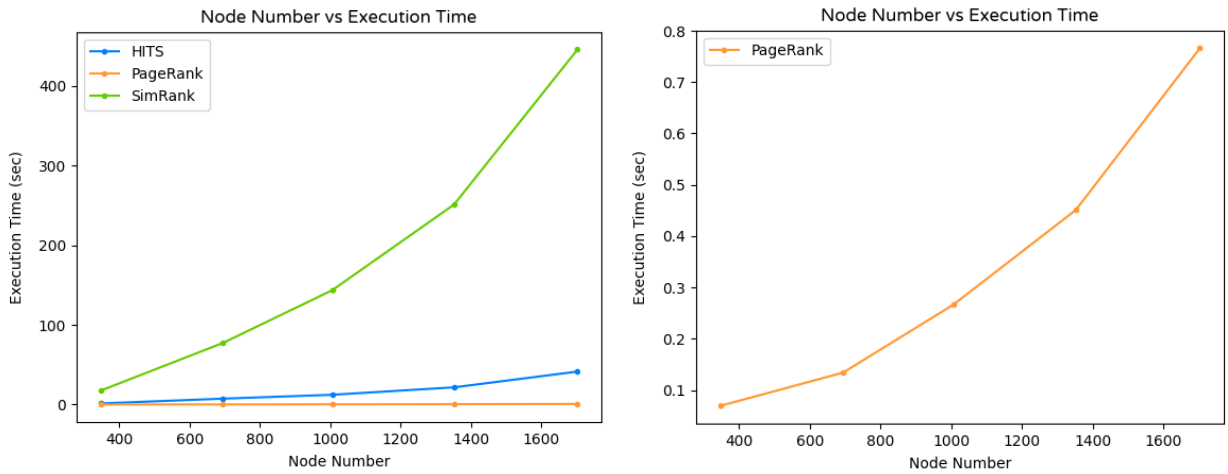

Figure 21. 不同 Node 數量對演算法執行時間的影響

PART 6 CONCLUSIONS

HITS

HITS 藉由計算 Authority 和 Hub 來評估每個 Node，並不會只因為 Parent 或 Children 數量多而就會有高的 Authority 和 Hub，Parent 和 Children 的品質也會影響到 Authority 和 Hub。

此外，在做這份作業時發生一件小插曲，也因此有個意外的發現。我跟同學在核對彼此 HITS 輸出的結果，但 Graph 6 都有對不上的地方，我同學跑 30 個 Iteration 得到的結果，我的程式碼要跑大概 60 個 Iteration 才能得到相似的結果。看了同學的程式碼之後才發現，我同學每一個 Iteration 在計算 Authority/Hub 時，都是用該 Iteration 剛計算完的 Hub/Authority 來計算，並不是用上一個 Iteration 的 Hub/Authority 來計算。也因此我才知道用這種方式計算的話，達到收斂的速度大概會是我的作法的兩倍(我達到收斂的 Iteration，我同學的只需要一半的 Iteration 就可達到收斂)。

PageRank

PageRank 則是藉由 Parent 的分數以及 Parent 的 Children 數量來做評分。基本上，PageRank 的分數大致上會與 Parent 數量呈一定相關性，因為 Parent 越多，就會有越多 PageRank 傳遞到該 Node。

若沒有使用 Damping Factor，則若 Graph 中存在沒有 Parent 的 Node，則該 Node 的 PageRank 就會是 0；倘若 Graph 存在沒有 Children 的 Node，則會有無解的狀況發生。

Damping Factor 越大，達到收斂的 Iteration 就會越少，這是因為「隨機」到達該 Node 的影響力大於從「Parent」到達該 Node 的影響力。此外，Damping Factor 越大，達到收斂時每個 Node 的 PageRank 就會越接近 1/(Node 總數)。

SimRank

SimRank 是藉由評估 Node 跟 Node 的 Parent 來評估這兩個 Node 的相似度。若某個 Node 的 Parent 是另一個 Node 的 Parent 的子集，則這兩個 Node 的 SimRank 就會比較高。此外，若兩個 Node 的 Parent 差集數量越小，則他們的 SimRank 也會較高。由此可知，若 Parent 的交集是空集合，那 SimRank 就會較低，不過在迭代的過程中，還會將 Parent 的 Parent 等關係較遠的 Parent 納入評估。

Decay Factor 越高，達到收斂的 Iteration 就會越多。但倘若達到收斂的 SimRank 是稀疏矩陣，或單位矩陣，則 Decay Factor 的大小對於達到收斂的 Iteration 的影響力則不大，因為這種狀況通常很快就會達到收斂。

Execution Time

在演算法的執行時間方面，三個演算法基本上都會隨著 Node 數量增加而增加執行時間，其中以 SimRank 的執行時間增加的幅度最為劇烈。在 Node 數量固定的情況下，HITS 和 PageRank 不太會因為 Edge 數量增加而顯著增加執行時間，但 SimRank 則會隨著 Edge 數量增加而顯著增加執行時間。三個演算法當中，PageRank 所需的執行時間是最少的。

最後，在實作演算法時，雖然呼叫 Package 的功能會很方便，但有可能會因此增加所需的執行時間，而誤判演算法的效能。