Data Mining Project 3 Report

H24116049 莊秉宸

A. HITS（Hyperlink Induced Topic Search）

● Algorithm description

HITS 藉由計算 Authority（權威值）和 Hubs（目錄值），決定網頁的重要程度。

- 若網頁具有 High Authority，代表它被許多具權威的網站所連
- 若網頁具有 High Hubs，代表它連到許多高權威性的網站

HITS 藉由遞迴運算 authority 和 hubs 並進行正規化，若兩者在迭代後的變化量很小，則視為收斂。HITS 的步驟如下：

1. 將 Authority 和 Hubs 初始化為 1
2. 每個節點的 Authority 為指向它的節點（母節點 parent node）Hubs 總和

$$A(node) = \sum_{w \in pn(node)} H(w)$$

3. 每個節點的 Hubs 為它指向節點（子節點 child node）的 Authority 總和

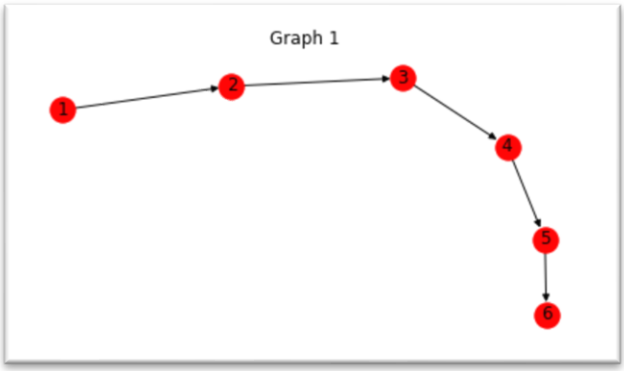
$$H(node) = \sum_{w \in cn(node)} A(w)$$

4. 將 authority 和 Hubs 進行正規化

$$for\ node\ in\ nodes:$$
$$A(node) = A(node) / \sum_{n \in nodes} A(n)$$
$$H(node) = H(node) / \sum_{n \in nodes} H(n)$$

5. 反覆迭代 2 至 4 步驟，直到收斂

- **Graph analysis**
  - **Graph1 - analysis**

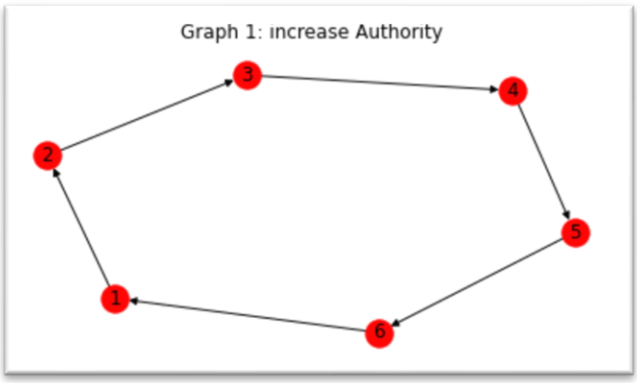


Authority: [0.0, 0.2, 0.2, 0.2, 0.2, 0.2]
Hub: [0.2, 0.2, 0.2, 0.2, 0.2, 0.0]

由於 node1 沒有指向它的母節點，因此它的 Authority 為 0；node6 沒有它指向的子節點，因此它的 Hubs 值為 0。

  - **Graph 1 – increase the Authority and Hub of node1**

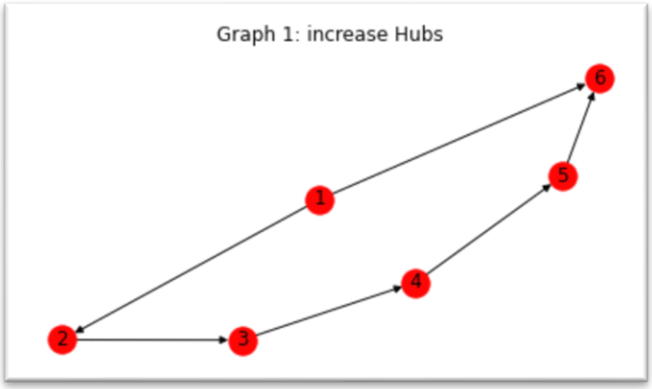
若要增加 node1 的 Authority，需要增加指向它的母節點。我設定 node6 為 node1 的母節點，建立一條 edge：



Authority: [0.167, 0.167, 0.167, 0.167, 0.167, 0.167]

我們可以發現 node1 的 Authority 從 0.0 -> 0.167。同理，將其餘 node 與 node1 建立 in-degree 關係，能使 node1 的 Authority 增加。

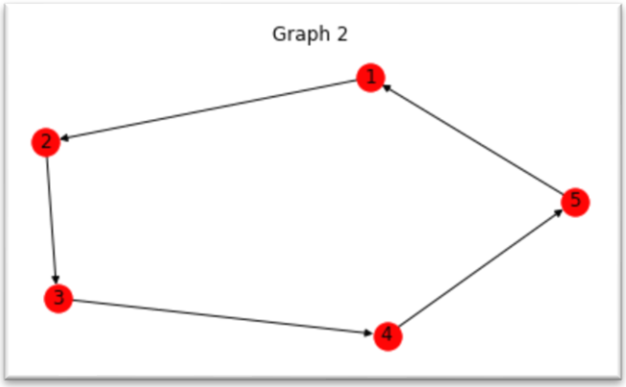
若要增加 node1 的 Hub 值，需要增加 node1 的 out-degree，我設定 node6 為 node1 的子節點，建立一條指向 node6 的 edge：



Hub: [0.618, 0.0, 0.0, 0.0, 0.382, 0.0]

我們可以發現，node1 的 Hub 從 0.2 -> 0.618。

➢ Graph2 – analysis

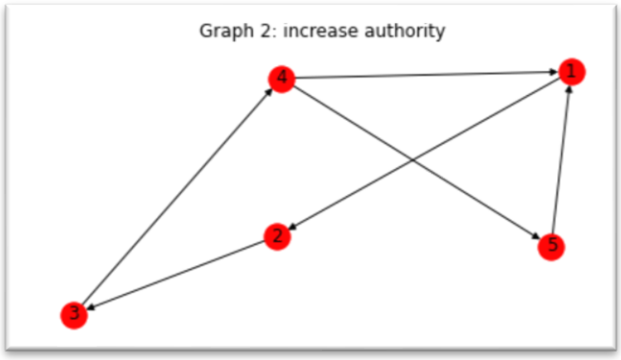


Authority: [0.2, 0.2, 0.2, 0.2, 0.2]
Hub: [0.2, 0.2, 0.2, 0.2, 0.2]

因為 Graph2 的節點呈現環狀，且每個節點都具有一個 parent node 和 child node，因此節點的 Authority 和 Hub 都一樣。

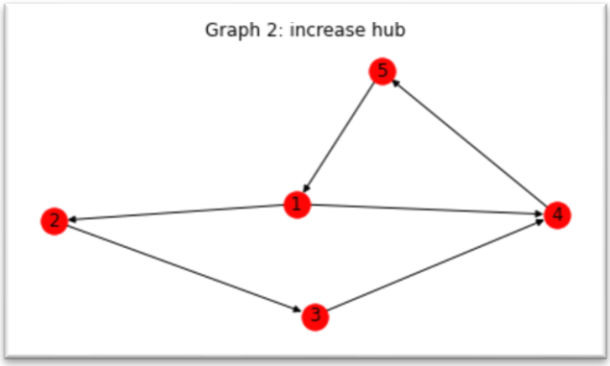
➢ Graph2 - increase the Authority and Hub of node1

為增加 node1 的 Authority，我在 node1 和 node4 間建立一條 in-degree 的 edge，將 node4 設定為 node1 的母節點：



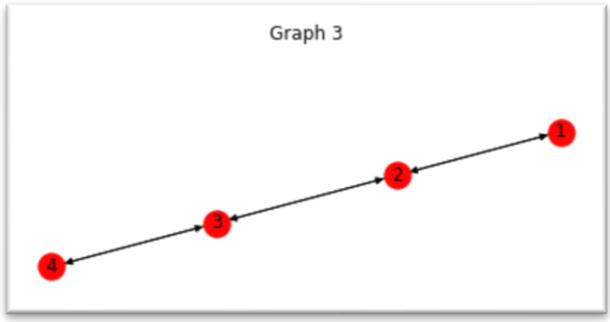
Authority: [0.618, 0.0, 0.0, 0.0, 0.382]
可以看見 node1 的 Authority 從 0.2 -> 0.618。

為增加 node1 的 Hub，我在 node1 和 node4 間建立一條 out-degree 的 edge，將 node4 設定為 node1 的子節點：



Hub: [0.618, 0.0, 0.382, 0.0, 0.0]
可以看見 node1 的 Hub 從 0.2 -> 0.618。

➢ Graph3 – analysis

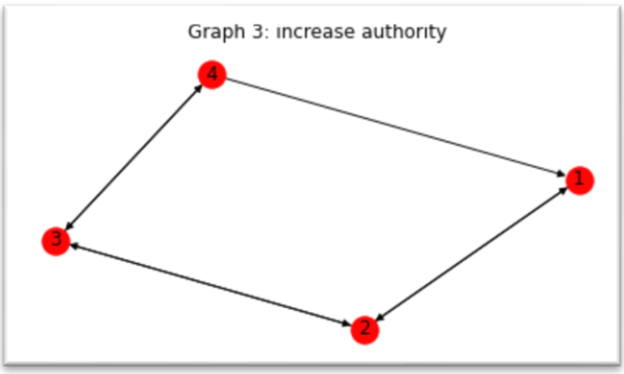


Authority: [0.191, 0.309, 0.309, 0.191]
Hub: [0.191, 0.309, 0.309, 0.191]

由於 node1、node4 具有各一個母節點和子節點，node2、node3 具有各兩個母節點和子節點，因此 2、3 節點相較 1、4 節點具有較高的 Authority 和 Hub。

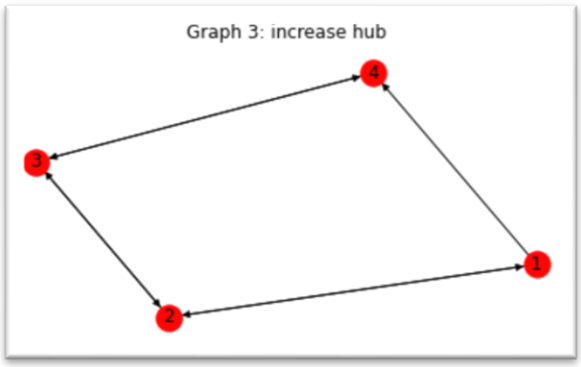
➢ Graph3 – increase the Authority and Hub of node1

為增加 node1 的 Authority，我在 node1 和 node4 增加了一條 in-degree 的 edge，將 node4 設定為 node1 的母節點：



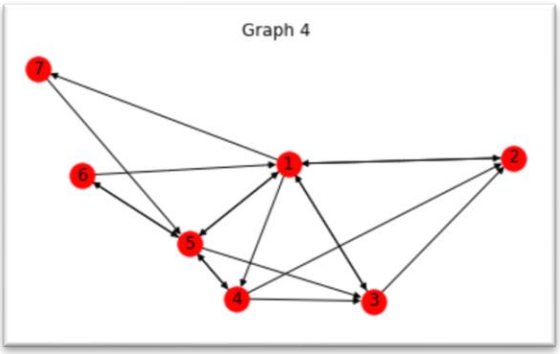
Authority: [0.5, 0.0, 0.5, 0.0]
可以發現 node1 的 Authority 從 0.191 -> 0.5。

為增加 node1 的 Hub，我在 node1 和 node4 增加了一條 out-degree 的 edge，將 node4 設定為 node1 的子節點：



Hub: [0.5, 0.0, 0.5, 0.0]
可以發現 node1 的 Hub 從 0.191 -> 0.5。

➢ Graph 4



Authority: [0.139, 0.178, 0.201, 0.14, 0.201, 0.056, 0.084]
Hub: [0.275, 0.048, 0.109, 0.199, 0.184, 0.117, 0.069]

➢ Graph 5, 6, ibm-5000 皆儲存在 result 資料夾中

B. PageRank

- ### Algorithm description

PageRank 演算法利用網站間的連結數，遞迴運算一個節點的 PageRank 值，評估網站的價值，具有高 PageRank 的網站，具有高的權威性。不同於 HIT 須建立 root set，且一個 set 只能對應一種 query，PageRank 不需建立 root set，且為 query independent。

- **若網頁被許多其他網頁連到，則有較高的 PageRank，比較重要**
- **若 PageRank 很高的網頁連到其他網頁，則連接的網頁 PageRank 值也會高**

PageRank 演算法的步驟如下：

1. **初始化節點的 PageRank 值，設定為 1/n （n 為節點數）**
2. **將節點 parent nodes 的 PageRank 除以其 out-degree edges 的數量，加總後搭配 damping factor（機率性的隨機跳出，預設為 0.1），計算出節點的 PageRank 值，公式如下：**

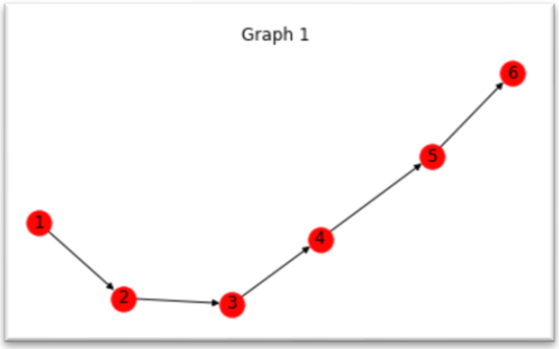
$$PR(P_i) = \frac{(d)}{n} + (1-d) \times \sum_{l_{j,i} \in E} PR(P_j) / \text{Outdegree}(P_j)$$

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

3. **將 PageRank 進行正規化。**
4. **反覆執行 2 和 3 步驟，直到收斂。**

- ### Graph analysis

  - #### Graph1 - analysis

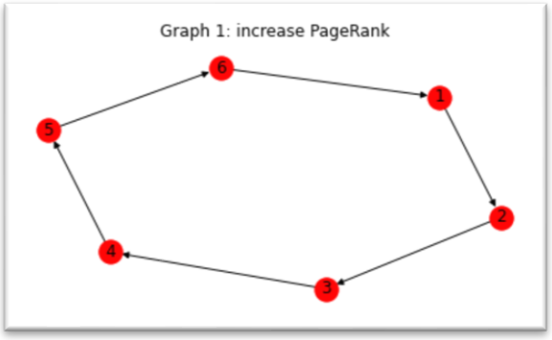


PageRank: [0.056, 0.107, 0.152, 0.193, 0.23, 0.263]

在 Graph1 中節點均為單向的情況下，PageRank 演算法會累加前面節點的 PageRank 值，因此越後面的節點 PageRank 越高。

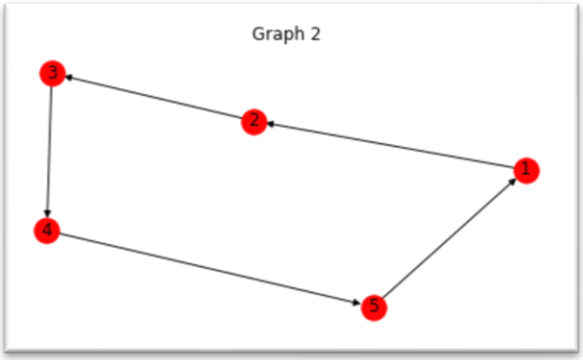
➢ Graph1 – increase pagerank of node1

我將 node6 連結 node1，形成 in-degree 的 edge，此時 node1 為 node6 的子節點。



PageRank: [0.167, 0.167, 0.167, 0.167, 0.167, 0.167]
可以發現 node1 的 PageRank 從 0.053 -> 0167。

➢ Graph2 - analysis

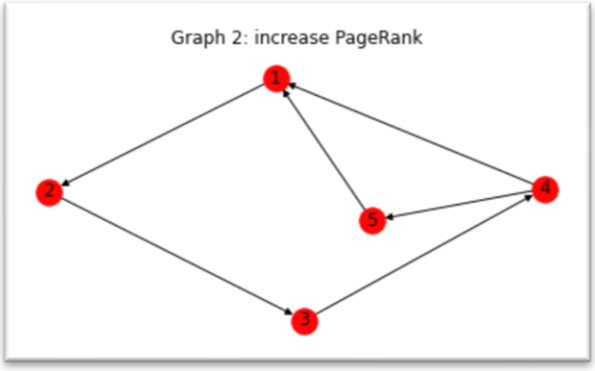


PageRank: [0.2, 0.2, 0.2, 0.2, 0.2]

由於 Graph2 為環狀結構，因此收斂後每個 node 的 PageRank 值相同。

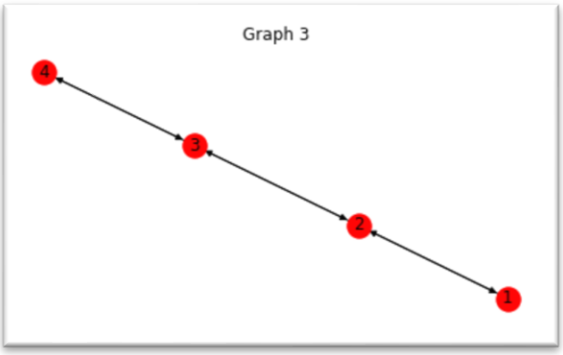
➢ Graph2 – increase pagerank of node1

我將 node4 連結 node1，形成 in-degree 的 edge，此時 node1 為 node4 的子節點。



PageRank: [0.224, 0.222, 0.219, 0.217, 0.118]
可以發現 node1 的 PageRank 從 0.2 -> 0.224。

➢ Graph3 - analysis

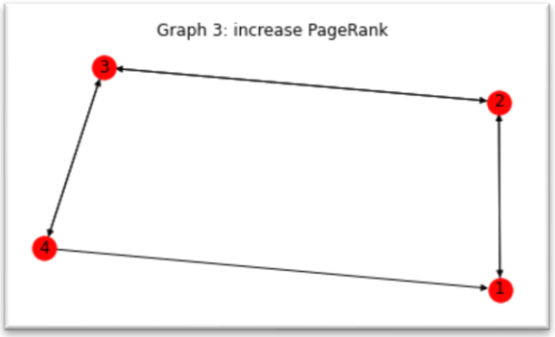


PageRank: [0.172, 0.328, 0.328, 0.172]

由於 node2、node3 有兩個母節點，而 node1、node4 只有一個，因此 node2、3 的 PageRank 較大。

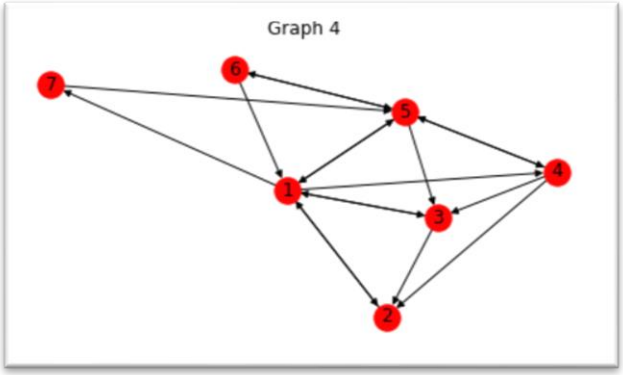
➢ Graph3 – increase pagerank of node1

我將 node4 連結 node1，形成 in-degree 的 edge，此時 node1 為 node4 的子節點。



PageRank: [0.25, 0.363, 0.25, 0.138]
可以發現 node1 的 PageRank 從 0.172 -> 0.25。

➢ Graph 4

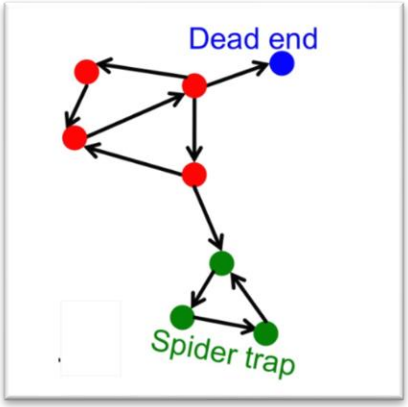


PageRank: [0.288, 0.161, 0.139, 0.107, 0.183, 0.055, 0.066]

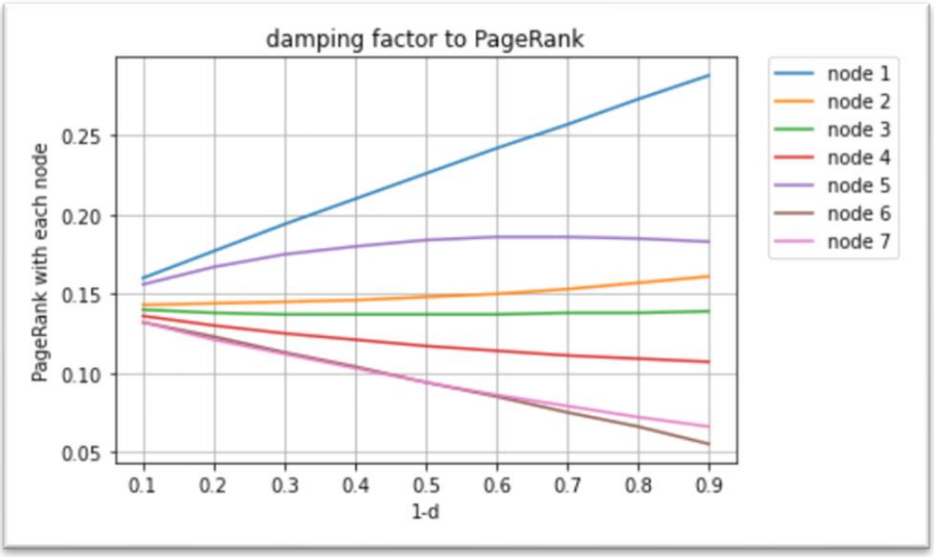
➢ Graph 5, 6, ibm-5000 皆儲存在 result 資料夾中

➢ Damping factor discussion

當圖中的部分節點形成一個環（如附圖）時，PageRank 演算法會困在環中，吸收掉其他節點的 PageRank 值。因此設定 damping factor，藉由隨機跳出環，避免 spider traps 的問題。



我使用 Graph4，搭配不同的 deamping_factor 值（0.1~0.9）並將每個節點計算出的 PageRank 值繪圖：



可以發現，當 d 值設定為 0.9 時，節點的 PageRank 差距較小，而當設 d=0.1 時，節點間的 PageRank 值差距變得更明顯，更容易對網頁進行優劣判斷。

C. SimRank

● Algorithm description

SimRank 的想法是，若兩個網站被類似的網站引用，則兩個網站間具有高相似度。藉由計算兩個節點母節點的相似程度，搭配 decay factor 計算出 SimRank 矩陣。

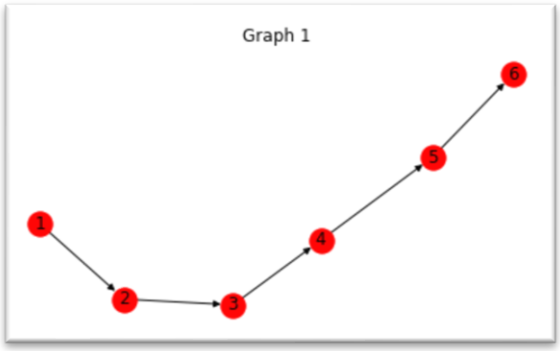
SimRank 演算法的步驟如下：

1. **初始化 SimRank 矩陣（n*n 矩陣，為 identity matrix）**
2. **計算節點間的相似度（節點 a, b 的母節點組合 SimRank 值的總和），乘上懲罰項，建立新的 SimRank 矩陣（若 a=b，回傳 SimRank=1，若 a, b 無母節點則回傳 0）。**
   C：decay_factor，預設為 0.7
   $I(a)$：節點 a 的母節點
   $I(b)$：節點 b 的母節點

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

3. **重複執行第二步，直到收斂**

● Graph 1

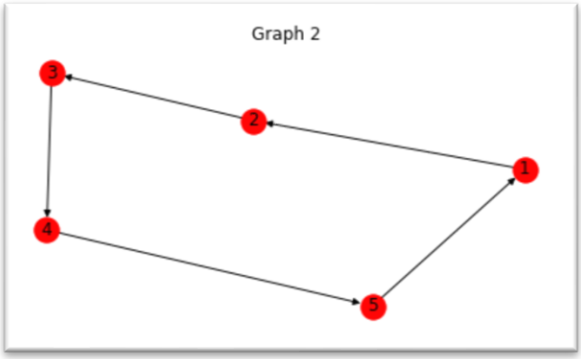


SimRank:

```
[[1.0, 0.0, 0.0, 0.0, 0.0, 0.0],
 [0.0, 1.0, 0.0, 0.0, 0.0, 0.0],
 [0.0, 0.0, 1.0, 0.0, 0.0, 0.0],
 [0.0, 0.0, 0.0, 1.0, 0.0, 0.0],
 [0.0, 0.0, 0.0, 0.0, 1.0, 0.0],
 [0.0, 0.0, 0.0, 0.0, 0.0, 1.0]]
```

由於兩兩節點間均無相同的母節點，因此相似度為 0，SimRank 矩陣為 identity matrix。

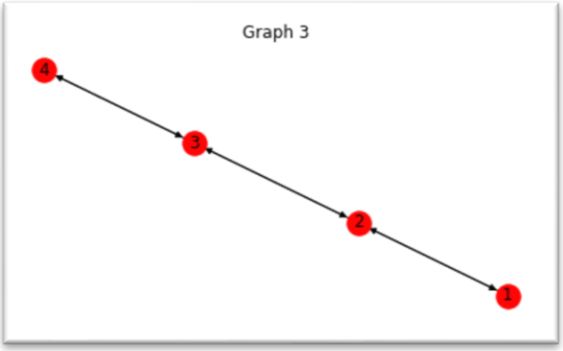
- ## Graph 2



SimRank:
```
[[1.0, 0.0, 0.0, 0.0, 0.0],
 [0.0, 1.0, 0.0, 0.0, 0.0],
 [0.0, 0.0, 1.0, 0.0, 0.0],
 [0.0, 0.0, 0.0, 1.0, 0.0],
 [0.0, 0.0, 0.0, 0.0, 1.0]]
```

由於兩兩節點間均無相同的母節點，因此相似度為 0，SimRank 矩陣為 identity matrix。

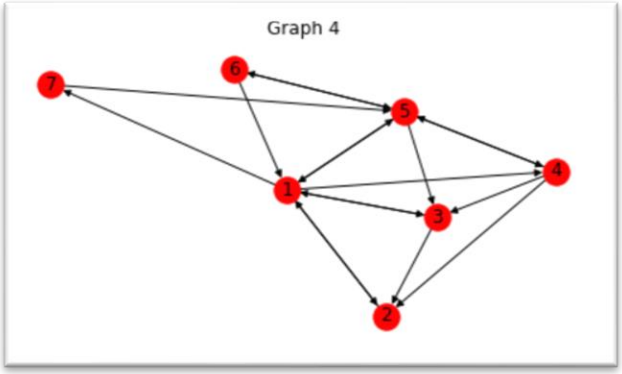
- ## Graph 3



SimRank:
```
[[1.0, 0.0, 0.538, 0.0],
 [0.0, 1.0, 0.0, 0.538],
 [0.538, 0.0, 1.0, 0.0],
 [0.0, 0.538, 0.0, 1.0]]
```

Node1 和 Node3 有相同的母節點 Node2
Node2 和 Node4 有相同的母節點 Node3
因此 SimRank[1][3]、SimRank[3][1]、SimRank[2][4]、SimRank[4][2] 均不為 0。

- ## Graph 4



SimRank:
```
[[1.0, 0.243, 0.232, 0.239, 0.221, 0.303, 0.175],
 [0.243, 1.0, 0.294, 0.256, 0.295, 0.17, 0.343],
 [0.232, 0.294, 1.0, 0.34, 0.275, 0.339, 0.341],
 [0.239, 0.256, 0.34, 1.0, 0.23, 0.427, 0.427],
 [0.221, 0.295, 0.275, 0.23, 1.0, 0.159, 0.3],
 [0.303, 0.17, 0.339, 0.427, 0.159, 1.0, 0.155],
 [0.175, 0.343, 0.341, 0.427, 0.3, 0.155, 1.0]]
```

- ## Graph 5 儲存在 result 資料夾中
- ## Decay factor discussion

如果一個網站連到非常多網站，這會導致子網站間具有高的 SimRank，然而他們之間不一定具有相似度，因此設立懲罰項 Decay factor，防止此現象。

我分別設置 decay factor 為 0.6, 0.7, 0.8，計算 Graph4 的 SimRank 矩陣。

- **Decay factor = 0.6**

```
[[1.0, 0.166, 0.158, 0.165, 0.148, 0.226, 0.105],
 [0.166, 1.0, 0.217, 0.183, 0.215, 0.101, 0.265],
 [0.158, 0.217, 1.0, 0.262, 0.199, 0.261, 0.263],
 [0.165, 0.183, 0.262, 1.0, 0.159, 0.344, 0.344],
 [0.148, 0.215, 0.199, 0.159, 1.0, 0.094, 0.224],
 [0.226, 0.101, 0.261, 0.344, 0.094, 1.0, 0.089],
 [0.105, 0.265, 0.263, 0.344, 0.224, 0.089, 1.0]]
```

- **Decay factor = 0.7**

```
[[1.0, 0.243, 0.232, 0.239, 0.221, 0.303, 0.175],
 [0.243, 1.0, 0.294, 0.256, 0.295, 0.17, 0.343],
 [0.232, 0.294, 1.0, 0.34, 0.275, 0.339, 0.341],
 [0.239, 0.256, 0.34, 1.0, 0.23, 0.427, 0.427],
 [0.221, 0.295, 0.275, 0.23, 1.0, 0.159, 0.3],
 [0.303, 0.17, 0.339, 0.427, 0.159, 1.0, 0.155],
 [0.175, 0.343, 0.341, 0.427, 0.3, 0.155, 1.0]]
```

- **Decay factor = 0.8**

```
[[1.0, 0.36, 0.349, 0.354, 0.338, 0.415, 0.292],
 [0.36, 1.0, 0.407, 0.37, 0.412, 0.285, 0.454],
 [0.349, 0.407, 1.0, 0.45, 0.39, 0.448, 0.451],
 [0.354, 0.37, 0.45, 1.0, 0.343, 0.535, 0.535],
 [0.338, 0.412, 0.39, 0.343, 1.0, 0.273, 0.412],
 [0.415, 0.285, 0.448, 0.535, 0.273, 1.0, 0.27],
 [0.292, 0.454, 0.451, 0.535, 0.412, 0.27, 1.0]]
```

可以發現，越高的 decay factor，節點的 SimRank 值會更高，同時母節點的數量也會

抑制 SimRank 的數值。

D. Effectiveness analysis

Graph name	HITS	PageRank	SimRank
Graph 1	7.97 ms	4.26 ms	8.68 ms
Graph 2	6.98 ms	6.66 ms	11.5 ms
Graph 3	7.57 ms	2.97 ms	3.63 ms
Graph 4	8.03 ms	5.21 ms	11.8 ms
Graph 5	3.36 s	2.62 s	470 s
Graph 6	53.3 s	36 s	x
Ibm-5000	33.2 s	16.2 s	x

觀察三種演算法於 Graph5 的計算效率，PageRank > HITS > SimRank，分析三種演算法的時間複雜度：

HITS 時間複雜度為 $O(E * I)$

- E：number of edges
- I：iterations

PageRank 時間複雜度為 $O(E * I)$

- E：number of edges
- I：iterations

SimRank 時間複雜度為 $O(In^2d)$

- I：iterations
- n：number of vertices
- d：average of product of parent nodes of pair of vertices

相較於 HITS 和 PageRank，能發現 SimRank 的時間複雜度很高，因此比其他兩種演算法花更多時間計算（470s）。

E. 小結

這份作業我實作了 HITS、PageRank 和 SimRank 三種 link analysis 的演算法，其實中心思想都不難，但 PageRank 卻能當作 Google 搜尋系統的排序方法，令我相當驚艷。

我在 HITS 和 PageRank 中均藉由操作 Graph 中結點的連接，達到增加 Hub, Authority 和 PageRank 值的目的，這使我對於演算法和數值的價值有了更透徹的了解，觀察不同 damping factor 對演算法的影響也讓我成功分析其對於 PageRank 演算法的價值。

我認為這份專案能有有效提升對 Link Analysis 的理解，藉由實作演算法外的討論和調整，對於三種演算法有更多的認識。