Data Mining 2022 Project3 Grading Policy v3

P76111262 林晨鈞

- Find a way
- Find a way (e.g., add/delete some links) to increase hub, authority, and PageRank of Node 1 in first 3 graphs respectively
  - graph_1
    - 原本:

      auth: {'1': 0.000, '2': 0.200, '3': 0.200, '4': 0.200, '5': 0.200, '6': 0.200}

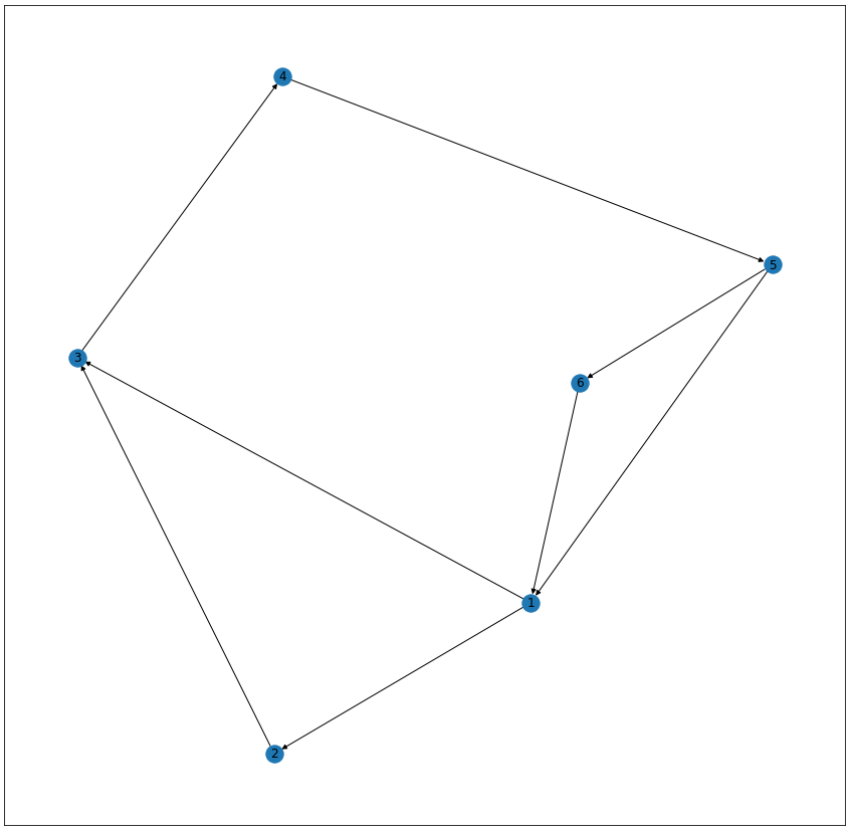
      hub: {'1': 0.200, '2': 0.200, '3': 0.200, '4': 0.200, '5': 0.200, '6': 0.000}

      pagerank: {'1': 0.017, '2': 0.032, '3': 0.045, '4': 0.057, '5': 0.068, '6': 0.078}
    - 增加(6,1)、(5,1)、(1,3)，可增加 Node 1 的 authority、hub:

      auth: {'1': 0.309, '2': 0.191, '3': 0.309, '4': 0.000, '5': 0.000, '6': 0.191}

      hub: {'1': 0.309, '2': 0.191, '3': 0.000, '4': 0.000, '5': 0.309, '6': 0.191}

      pagerank: {'1': 0.198, '2': 0.106, '3': 0.201, '4': 0.197, '5': 0.194, '6': 0.104}



  - graph_2
    - 原本:

      auth: {'1': 0.200, '2': 0.200, '3': 0.200, '4': 0.200, '5': 0.200,}

      hub: {'1': 0.200, '2': 0.200, '3': 0.200, '4': 0.200, '5': 0.200}

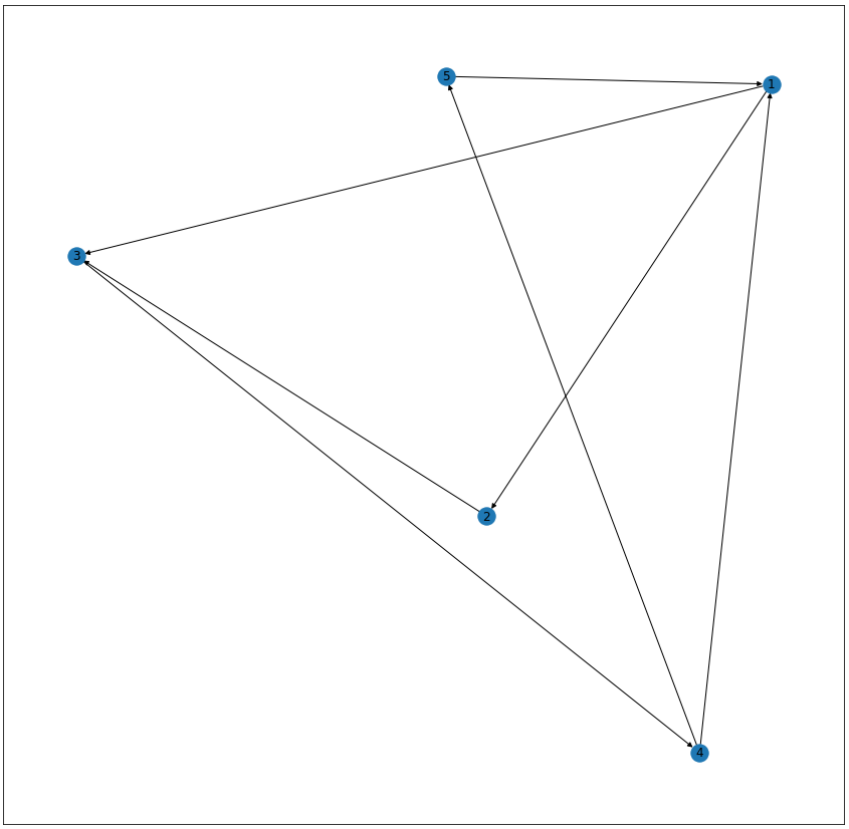
pagerank: {'1': 0.200, '2': 0.200, '3': 0.200, '4': 0.200, '5': 0.200}

- 增加(1,3)、(2,3)、(4,1)，可增加 Node 1 的 authority、hub:

auth: {'1': 0.309, '2': 0.191, '3': 0.309, '4': 0.000, '5': 0.191}

hub: {'1': 0.309, '2': 0.191, '3': 0.000, '4': 0.309, '5': 0.191}

pagerank: {'1': 0.247, '2': 0.131, '3': 0.249, '4': 0.244, '5': 0.130}



- graph_3
  - 原本:

auth: {'1': 0.191, '2': 0.309, '3': 0.309, '4': 0.191}

hub: {'1': 0.191, '2': 0.309, '3': 0.309, '4': 0.191}

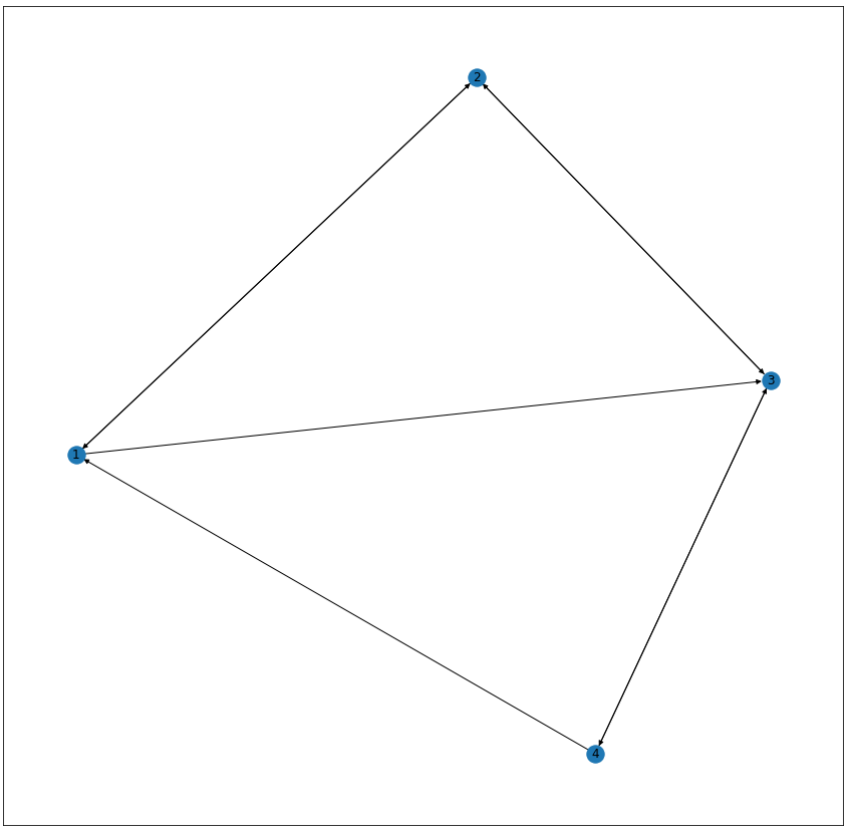
pagerank: {'1': 0.172, '2': 0.328, '3': 0.328, '4': 0.172}

- 增加(1,3)、(2,3)、(4,1)，可增加 Node 1 的 authority、hub:

auth: {'1': 0.322, '2': 0.178, '3': 0.453, '4': 0.047}

hub: {'1': 0.262, '2': 0.322, '3': 0.093, '4': 0.322}

pagerank: {'1': 0.226, '2': 0.274, '3': 0.328, '4': 0.172}



- Algorithm description
- 解釋每個演算法步驟流程
  - HITS(Hypertext Induced Topic Selection)
    1. 初始化所有節點的 hub 和 auth 值為 1
    2. 在每個迭代中，更新每個節點的 hub 和 auth 值
    3. 新的 authority 值的計算方法是該節點的所有父親(指向自己的節點)節點的 hub 值的加總
    4. 新的 hub 值的計算方法是該節點的所有兒子(自己指出去指到的點)節點的 authority 值的加總
    5. 對所有新計算出的 authority 和 hub 值做正規化

```python
def Hits(graph,iter):

    hubs = dict.fromkeys(graph,1) # 每個點的hub值都設1
    auth = dict.fromkeys(graph,1) # 每個點的auth值都設1

    for i in range(iter):
        lastHubs = hubs # 儲存上一個圖的hub值
        lastAuth = auth # 儲存上一個圖的auth值
        hubs = dict.fromkeys(lastHubs.keys(), 0)
        auth = dict.fromkeys(lastHubs.keys(), 0)
        for node in auth: # node 是 1 2 3 4 5 6
            for predecessors in graph.predecessors(node): # 找出節點的父親(所有指向自己的節點)
                auth[node] += lastHubs[predecessors] # 新的auth值為之前的父親的hub加總
        for node in hubs:
            for successors in graph.successors(node): # 找出節點的兒子(所有自己指出去指到的點)
                hubs[node] += lastAuth[successors] # 新的hub值為之前的兒子的auth加總

        auth_sum = 0
        hubs_sum = 0
        for value in auth.values():
            auth_sum += abs(value)
        for value in hubs.values():
            hubs_sum += abs(value)
        # 對auth和hub做正規化
        for key in auth:
            auth[key] = auth[key]/auth_sum
        for key in hubs:
            hubs[key] = hubs[key]/hubs_sum

    return auth, hubs
```

- PageRank

1. 初始化所有節點的 PR 值為(1/節點總數)
2. 接著使用以下公式迭代更新各節點 PR 值

   d: damping factor

   n: 節點總數

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

```python
def PageRank(graph, iter, damping):
    n = graph.number_of_nodes() # 計算圖中的點的個數
    d_frac = damping / n
    rank = dict.fromkeys(graph, 1.0 / n) # 每個節點賦值 1/n
    for _ in range(iter):
        lastRank = rank
        for node in rank: # 全部的點跑一次
            rankSum = 0
            for predecessors in graph.predecessors(node): # 找出節點的父親(所有指向自己的節點)
                outdegree = graph.out_degree(predecessors) # 計算該節點的outdegree值
                if outdegree > 0:
                    rankSum += lastRank[predecessors] * (1.0 / outdegree) # 依據公式計算
            rank[node] = d_frac + ((1 - damping) * rankSum) # 依據公式計算
    return rank
```

- SimRank

1. 首先，為整張圖儲存一個 n*n 的單位矩陣
2. 並根據以下公式進行迭代，以更新點對點的 simrank 值

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

3. S(a,b)依據以下規則更新: (a,b 為圖中的兩個節點)

   If (a == b):

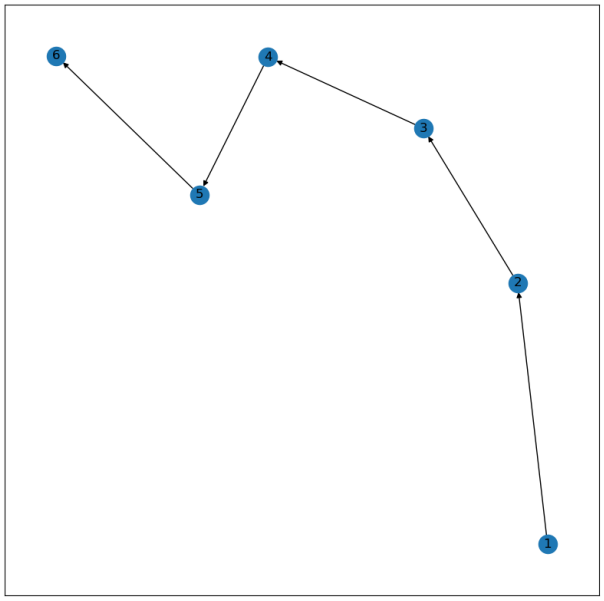
   Simrank(a,b) = 1

   Else:

   Simrank(a,b) = 0
4. I(a): 節點 a 的所有父親節點(所有指向自己的節點)
5. C: decay factor

```python
def simrank(graph, c, iter):
    nodenum = graph.number_of_nodes()
    srank = [[0]*nodenum for i in range(nodenum)]
    for i in range(nodenum): # 首先先建立第一張表
        for j in range(nodenum):
            row = i + 1
            col = j + 1
            row = str(row)
            col = str(col)
            if i == j: # 矩陣的對角線都設1
                srank[i][i] = 1
            else:
                Sum = 0
                row_parent_num = graph.in_degree(row) # 計算 node1 的父親數量
                col_parent_num = graph.in_degree(col) # 計算 node2 的父親數量
                for r_predecessors in graph.predecessors(row): # 針對所有 node1 的父親迴圈(排列組合)
                    for c_predecessors in graph.predecessors(col): # 針對所有 node2 的父親迴圈(排列組合)
                        if r_predecessors == c_predecessors: # 若 node1 = node2，總和+1
                            Sum += 1
                if row_parent_num == 0 or col_parent_num == 0: # 若 node1 或 node2 沒有父親，設0
                    srank[i][j] = 0
                else: # 依據公式
                    srank[i][j] = c * (1.0/(row_parent_num*col_parent_num)) * Sum
    for k in range(iter-1): # 進行迭代
        lastsrank = srank # 將上個迭代的表存起來
        for i in range(nodenum):
            for j in range(nodenum):
                row = i + 1
                col = j + 1
                row = str(row)
                col = str(col)
                row_parent_num = graph.in_degree(row) # 計算 node1 的父親數量
                col_parent_num = graph.in_degree(col) # 計算 node2 的父親數量
                if i == j:
                    continue
                else:
                    Sum = 0
                    for r_predecessors in graph.predecessors(row): # 針對所有 node1 的父親迴圈(排列組合)
                        for c_predecessors in graph.predecessors(col): # 針對所有 node2 的父親迴圈(排列組合)
                            Sum += lastsrank[int(r_predecessors)-1][int(c_predecessors)-1] # 查詢上個迭代的表，並進行加總
                    if row_parent_num == 0 or col_parent_num == 0: # 若 node1 或 node2 沒有父親，設0
                        srank[i][j] = 0
                    else: # 依據公式
                        srank[i][j] = c * (1.0/(row_parent_num*col_parent_num)) * Sum
    return srank
```

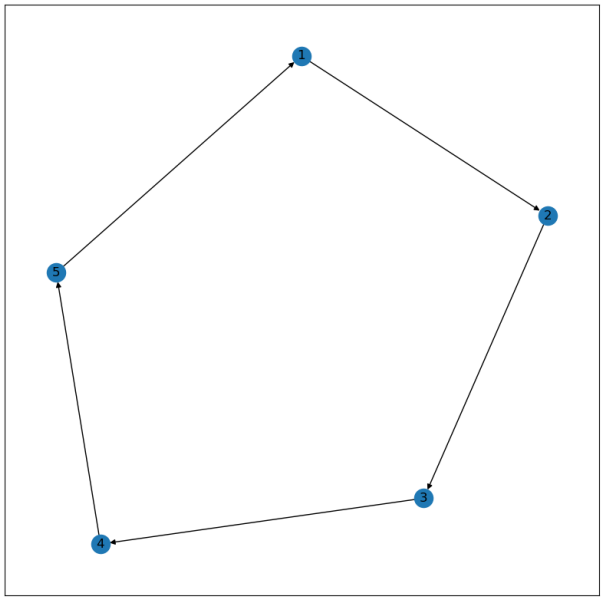
- Result Analysis and Discussion
  - graph_1



    - HITS:
      - ✓ Authority: 0.000 0.200 0.200 0.200 0.200 0.200
      - ✓ Hub: 0.200 0.200 0.200 0.200 0.200 0.000
      - ✓ auth 值的大小取決於有多少 Node 指向自己，hub 值取決於自己指向多少個 Node。
      - ✓ Node1 的 authority 值為 0，因為沒有任何 Node 指向 Node1。
      - ✓ Node6 的 hub 值為 0，因為 Node6 沒有指向任何點。
    - PageRank(PR):
      - ✓ 0.017 0.032 0.045 0.057 0.068 0.078
      - ✓ 因為此圖是一個單向流的圖，而 pagerank 的算法，Node2 的 PR 值會加上加權後 Node1 的 PR 值，Node3 的 PR 值會加上加權後 Node2 的 PR 值，以此類推，所以 Node1 到 Node6 的 PR 值是由小到大。
    - SimRank:
      - ✓ 1.000 0.000 0.000 0.000 0.000 0.000
        0.000 1.000 0.000 0.000 0.000 0.000
        0.000 0.000 1.000 0.000 0.000 0.000
        0.000 0.000 0.000 1.000 0.000 0.000
        0.000 0.000 0.000 0.000 1.000 0.000
        0.000 0.000 0.000 0.000 0.000 1.000
      - ✓ Simrank 的計算中，兩個節點是否有相同父親是非常重要的，此圖中任兩節點都不存在相同父親，所以除了對角線外，其

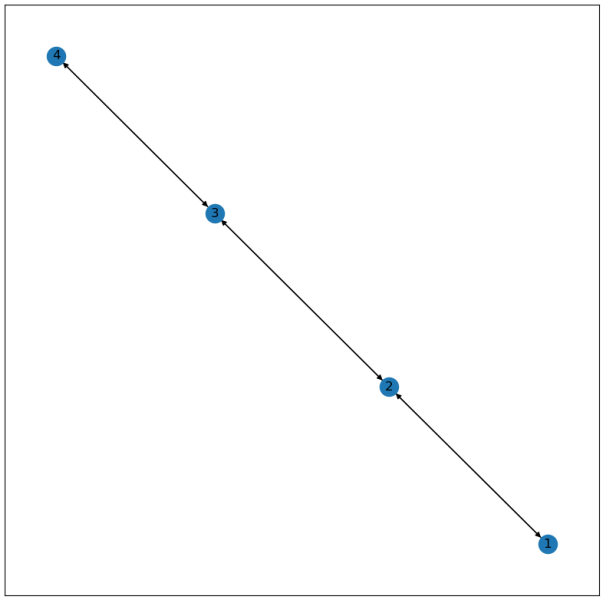
餘的值都是 0。

- graph_2



  - HITS:
    - Authority: 0.200 0.200 0.200 0.200 0.200
    - Hub: 0.200 0.200 0.200 0.200 0.200
    - 此圖的所有節點連結形成一個迴圈，每個節點都有一個父親和兒子，所以每個節點的 auth 和 hub 值理所當然都是相同的。
  - PageRank(PR):
    - 0.200 0.200 0.200 0.200 0.200
    - 此圖的所有節點連結形成一個迴圈，每個節點的 PR 值會收斂為相同的值。
  - SimRank:
    - 1.000 0.000 0.000 0.000 0.000
      0.000 1.000 0.000 0.000 0.000
      0.000 0.000 1.000 0.000 0.000
      0.000 0.000 0.000 1.000 0.000
      0.000 0.000 0.000 0.000 1.000
    - Simrank 的計算中，兩個節點是否有相同父親是非常重要的，此圖中任兩節點都不存在相同父親，所以除了對角線外，其餘的值都是 0。

- graph_3



- HITS:
  - ✓ Authority: 0.191 0.309 0.309 0.191
  - ✓ Hub: 0.191 0.309 0.309 0.191
  - ✓ Node2 和 Node3 都有 2 個父親和 2 個兒子，所以 auth 和 hub 值會較高。
  - ✓ Node1 和 Node4 則都只有 1 個父親和 1 個兒子，所以 auth 和 hub 值會較小。
- PageRank:
  - ✓ 0.172 0.328 0.328 0.172
  - ✓ Node2 和 Node3 都有 2 個父親和 2 個兒子，所以 PR 值會較高。
  - ✓ Node1 和 Node4 則都只有 1 個父親和 1 個兒子，所以 PR 值會較小。
- SimRank:
  - ✓ 1.000 0.000 0.538 0.000
    0.000 1.000 0.000 0.538
    0.538 0.000 1.000 0.000
    0.000 0.538 0.000 1.000
  - ✓ 可以觀察到 Node pair(1,3)、pair(1,4)都有各自共同的父親，所以 simrank 值不為 0。

■ 針對不同 damping factor 和 decay factor 進行討論

- damping factor
  - ✓ 當 damping factor 趨近於 1 時，會導致所有節點的 pagerank 值均勻分布。
  - ✓ 當 damping factor 趨近於 0 時，有最多父親的節點會有較大

的可能得到最大 pagerank 值。

- ✓ graph_4 搭配 damping factor = 0.1
  - ◆ 0.289 0.161 0.140 0.107 0.183 0.055 0.066
- ✓ graph_4 搭配 damping factor = 0.9
  - ◆ 可看出每個節點的 pagerank 平均分布
  - ◆ 0.160 0.143 0.140 0.136 0.156 0.132 0.132

➢ decay factor

- ✓ 因為相同 node 之間的 simrank 值是 1，也就是 s(x,x) = 1，所以為了區分極高相似與完全相同之間的差異，也就是避免 s(a,b) = 1，所以引進 decay factor。
- ✓ graph_4 搭配 decay factor = 0.7
  - ◆ 1.000 0.243 0.232 0.239 0.221 0.303 0.175
    0.243 1.000 0.294 0.256 0.295 0.170 0.343
    0.232 0.294 1.000 0.340 0.275 0.339 0.341
    0.239 0.256 0.340 1.000 0.230 0.427 0.427
    0.221 0.295 0.275 0.230 1.000 0.159 0.300
    0.303 0.170 0.339 0.427 0.159 1.000 0.155
    0.175 0.343 0.341 0.427 0.300 0.155 1.000
- ✓ graph_4 搭配 decay factor = 0.1
  - ◆ 1.000 0.010 0.010 0.014 0.008 0.026 0.001
    0.010 1.000 0.023 0.018 0.019 0.001 0.034
    0.010 0.023 1.000 0.034 0.018 0.034 0.034
    0.014 0.018 0.034 1.000 0.014 0.050 0.050
    0.008 0.019 0.018 0.014 1.000 0.001 0.026
    0.026 0.001 0.034 0.050 0.001 1.000 0.001
    0.001 0.034 0.034 0.050 0.026 0.001 1.000
- ✓ graph_4 搭配 decay factor = 0.99
  - ◆ 1.000 0.934 0.933 0.933 0.931 0.941 0.925
    0.934 1.000 0.940 0.935 0.941 0.924 0.946
    0.933 0.940 1.000 0.945 0.938 0.944 0.945
    0.933 0.935 0.945 1.000 0.931 0.956 0.956
    0.931 0.941 0.938 0.931 1.000 0.922 0.940
    0.941 0.924 0.944 0.956 0.922 1.000 0.922
    0.925 0.946 0.945 0.956 0.940 0.922 1.000
- ✓ 可以觀察到 decay factor 設太小或太大的話，會導致兩個不同節點的 simrank 值很小或很大，看不出差異，所以 simrank 值的設定是很重要的。

- Effectiveness analysis

PageRank

Graph	Time (seconds)
graph_1	0.0
graph_2	0.0
graph_3	0.0
graph_4	0.0
graph_5	0.024915695190429688
graph_6	0.11833882331848145
IBM_dataset	0.0547480583190918

HITS

Graph	Time (seconds)
graph_1	0.0
graph_2	0.0
graph_3	0.0
graph_4	0.0
graph_5	0.014954090118408203
graph_6	0.05481982231140137
IBM_dataset	0.029900074005126953

SimRank

Graph	Time (seconds)
graph_1	0.0029904842376708984
graph_2	0.0029883384704589844
graph_3	0.0019958019256591797
graph_4	0.007990837097167969
graph_5	22.085312366485596
graph_6	314.83754229545593
IBM_dataset	

- 由上表可以得知，在 graph1~graph4 這些節點及邊數較少的圖，3 個不同的演算法的執行時間都很少，但當遇到 graph5、graph6、IBM_dataset 這種節點跟邊數都非常多的圖來說，執行時間就會更多，可以看到 SimRank 的執行時間又更多得原因是在於要去一一比對兩兩節點間是否存在相同的父親，這又使得 SimRank 這個演算法要花額外的時間去計算。