Datamining Project 3 P86114165

HackMD_page (https://hackmd.io/@ohYF12gcROi7Ad_XDuEvvw/HyeM8g3L6)

參數設定

- damping_factor = 0.1
- decay_factor = 0.7
- iteration = 30
- authority, hub 初始值：1
- pagerank 初始為 1/N

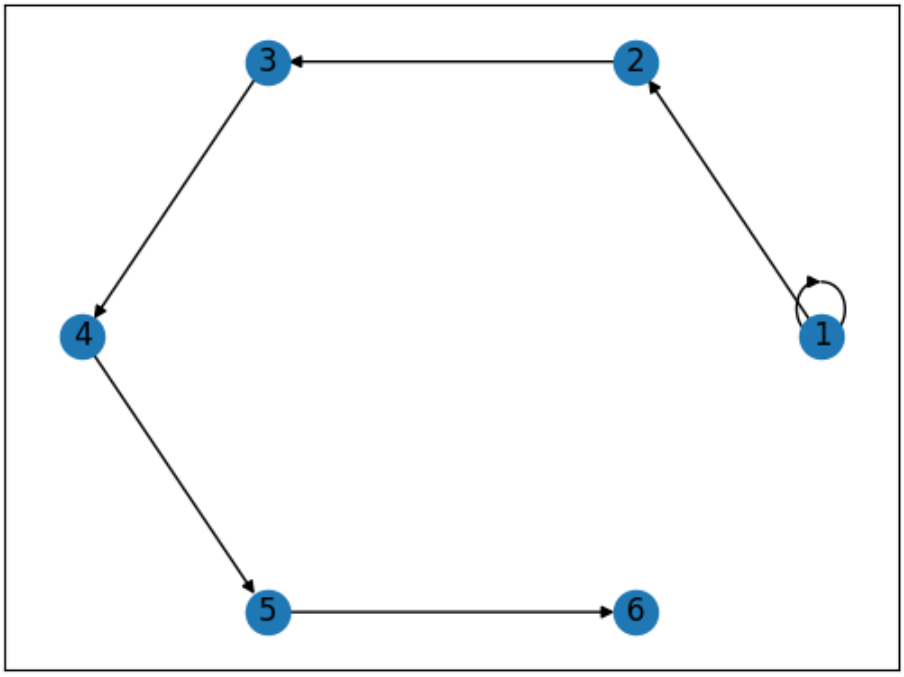
Find a way (提高 authority, hub, pagerank)

Graph_1

- 原始輸出: authority: 0, hub: 0.2, pagerank=0.017
- 將 1 自己 link 到自己
- 更正後輸出: authority: 0.5, hub: 1, pagerank: 0.030
- 增加 1,1 link 的原因是因為想要增加入度 (沒有人連到 node 1)，又因為若是其他點連到 node 1 的話可能會減少 ｈｕｂ （例如 node 6 如果連到 node 1 形成閉環, authority 和 hub 都會變成 0.167), 所以讓自己連到自己，確保 authority 增加hub 也一定增加

```
Authority: [0.5 0.5 0.  0.  0.  0. ]
Hub: [1. 0. 0. 0. 0. 0.]
PageRank: [0.03  0.03  0.044 0.056 0.067 0.077]
```

graph_1 更正後結果



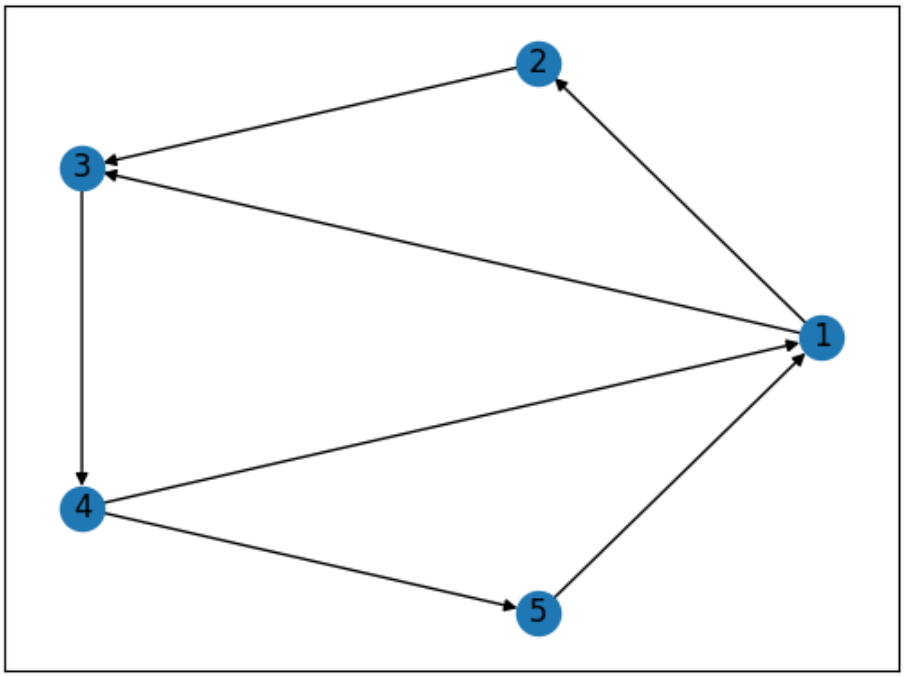
graph_1 增加 link 後

Graph_2

- 原始輸出: authority: 0.2 , hub: 0.2, pagerank=0.2
- 增加 1 到 3 的 link 和 4 到 1 的 LINK
- 更正後輸出: authority: 0.309, hub: 0.309, pagerank: 0.247
- 因為原本 graph_2 是一個閉環, 大家的的數值都一樣所以增加 node 1 的重要性 (增加入度與出度)

```
Authority: [0.309 0.191 0.309 0.    0.191]
Hub: [0.309 0.191 0.    0.309 0.191]
PageRank: [0.247 0.131 0.249 0.244 0.13 ]
```

graph_2 更正後結果



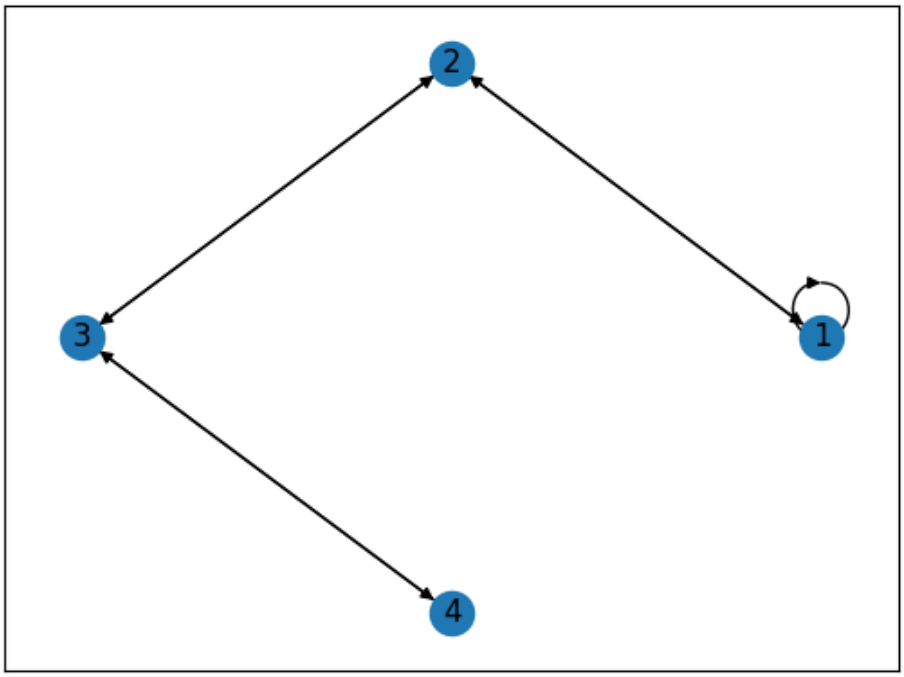
graph_2 增加 link 後

Graph_3

- 原始輸出: authority: 0.191 , hub: 0.191, pagerank=0.172
- 增加 1到自己的 link
- 更正後輸出: authority: 0.347, hub: 0.347, pagerank: 0.274
- 思路跟　graph_1 很像，增加入度，又因為自己的 authority 增加 指自己也能讓 hub 增加

```
Authority: [0.347 0.305 0.227 0.121]
Hub: [0.347 0.305 0.227 0.121]
PageRank: [0.274 0.279 0.291 0.156]
```

graph_3 更正後結果



graph_3 增加 link 後

Algorithm description

圖形輸入 (轉化為 Adjacency Matrix)

read file

```
def read_file(filename: Union[str, Path],arg) -> List[List[int]]:
    if arg.dataset=='ibm-5000.txt': # 如果輸入是 ibm-5000 的資料處理方式不同
        #逐行讀入資料
        file_temp=[
            [x for x in line.split()]
            for line in Path(filename).read_text().splitlines()]
        result=[]
        #每行包含三個數字 例如第一行 line 就會等於 ['1', '1', '307']
        for line in file_temp:
            temp=[]
            num=0
            for word in line:
                if num!=1:# 三個數字只取頭尾兩個為邊的起終點
                    temp.append(int(word))#將編號轉為 integer 方便之後對節點做排序
                num+=1
            result.append(temp.copy())#回傳資料 (邊的資訊，且為 integer 的 list List[Li

    else:# ibm之外的資料處理方式
        #逐行讀入資料
        file_temp=[
            [x for x in line.split()]
            for line in Path(filename).read_text().splitlines()]
        result=[]
        for line in file_temp: # 每行資料為一個字串陣列 例如 graph_1 第一行['1,2']
            temp=[]
            word_temp=""#暫放數字的變數
            for list_line in line:#list_line 是字串，因為 line 是字串陣列，例如 graph_1
                #收集數字, 以 ',' 為分界，讀到 ',' 就將 world_temp 的字串轉為 integer
                for word in list_line:
                    if word ==',':
                        temp.append(int(word_temp))#將編號轉為 integer 方便之後對節點做
                        word_temp=""

                    else:
                    #若為十位數字以上就會以字串形式收集在 word_temp中，例如 307 就會以 '3'
                        word_temp=word_temp+word
                temp.append(int(word_temp))#終點後沒有 ',', 所以需要再 append 一次
                result.append(temp.copy())#回傳資料 (邊的資訊，且為 integer 的 list Lis
    return result.copy()
```

建構圖

```
# 將input data 存成一張圖，以供後續應用

class vertex:
    '''
    節點類型
    提供圖的最基礎架構
    包括節點名稱
    子節點名稱 ()
    父節點名稱 ()
    '''
    def __init__(self,name) -> None:
        self.name=name
        self.children_v={}
        self.parent_v={}

class DAG:
    '''
    圖類型

    '''

    def __init__(self) -> None:
        '''
        透過 add_vertex 方法 將 input data 存成圖
        並同步以 networkx 方式存有向圖以供視覺化
        '''
        self.vertex_list={}
        self.G=nx.DiGraph()

    def add_vertex(self,name):
        '''
        透過 add_vertex 方法 將 input data 存成圖
        並同步以 networkx 方式存有向圖以供視覺化
        '''
        self.vertex_list[name]=vertex(name)

    def add_edge(self,parent,children):
        '''
        輸入起終點便能建立邊：
        在起點的子節點中加入終點，終點的父節點中加入起點。
        '''
        self.G.add_edge(parent,children)
        if parent not in self.vertex_list.keys():
            self.add_vertex(parent)
        if children not in self.vertex_list.keys():
            self.add_vertex(children)
        self.vertex_list[parent].children_v[children]=1
        self.vertex_list[children].parent_v[parent]=1

    def visualize(self):
        '''
        使用 networkx 視覺化有向圖
        '''
        pos=nx.circular_layout(self.G)
        nx.draw_networkx(self.G,pos)
        plt.show()

    def visual_table(self):
        '''
```

```
        列出所有邊
        '''

        for ind in self.vertex_name():
            for ind2 in self.vertex_list[ind].children_v.keys():
                print("[",ind,",",ind2,"]")
    def vertex_num(self):
        '''
        回傳圖中點數量
        '''
        return len(self.vertex_list.keys())
    def vertex_name(self):
        '''
        回傳排序好的圖的節點編號
        '''
        myKeys = list(self.vertex_list.keys())
        myKeys.sort()
        return myKeys
```

前處理

```

def Adjacent_matrix(graph:DAG):
    '''
    將已存成 DAG 類型的圖存成鄰接矩陣
    '''
    length=graph.vertex_num()
    A_M=np.zeros([length,length])
    ind_dict={}#將圖的點編號以 dictionary 對應到矩陣 index，例如有 [1,2,4,5] 四個點， ind
    # 將排序後的圖得節點編號對應到矩陣的 ｉｎｄｅｘ
    for ind,name in enumerate(graph.vertex_name()):
        ind_dict[name]=ind
    #有 a 到 b 的邊的話，將鄰接矩陣 A_M 的 [a,b] 設為 1
    for ind in graph.vertex_name():
            for ind2 in graph.vertex_list[ind].children_v.keys():
                A_M[ind_dict[ind]][ind_dict[ind2]]+=1
    return A_M

def to_graph(input_data):
    '''
    將輸入資料存成 DAG類型
    '''
    graph=DAG()
    for i in input_data:
        graph.add_edge(i[0],i[1])
    return graph

```

HITS

演算法參照 (老師講義第 19 頁)：

Basic Link Analysis

- Let A denote the **adjacency matrix** of the graph, $\boldsymbol{a}_t \leftarrow A^t\boldsymbol{h}_{t-1}$, $\boldsymbol{h}_t \leftarrow A\,\boldsymbol{a}_{t-1}$
  - $\boldsymbol{a}_n$ is the unit vector in the direction of $(A^tA)^{n-1}A^tz$
  - $\boldsymbol{h}_n$ is the unit vector in the direction of $(AA^t)^nz$
- $\boldsymbol{a}^*$ is the principal eigenvector of A^tA, and $\boldsymbol{h}^*$ is the principal eigenvector of AA^t

```
1
2  def self_HITS(input_data,arg):
3      '''
4      實作 HITS 演算法
5      '''
6      start_time=time.time()
7      #前處理###################################
8      graph_1=to_graph(input_data)
9      A_M=Adjacent_matrix(graph_1)
10     ##########################################
11
12     G_length=graph_1.vertex_num()#取得 N
13     au=np.ones([G_length])#初始化 authority 矩陣為 1
14     hu=np.ones([G_length])#初始化 hub 矩陣為 1
15
16     for itr in range(arg.itr):
17         au=np.matmul(A_M.T,hu)#a_t=(A^T)(h_t-1)
18         hu=np.matmul(A_M,au)#h_t=A(a_t-1)
19         au=au/np.sum(au)#normalized
20         hu=hu/np.sum(hu)#normalized
21     end_time=time.time()
22     if arg.print_result:#如果要印出 result 可以將  --print_result 設為 True
23         print("Authority:",au)
24         print("Hub:",hu)
25     if arg.show_time:#如果要印出 計算時間 可以將  --show_time 設為 True
26         print("HITS_Computation_time:",end_time-start_time)
27     return au,hu
```

PageRank

演算法參照 (老師講義第 37 頁, WIKI) :

Quick reference

37

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

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

Iterative [edit]

At $t = 0$, an initial probability distribution is assumed, usually

$$PR(p_i;0) = \frac{1}{N}.$$

where N is the total number of pages, and $p_i;0$ is page i at time 0.

At each time step, the computation, as detailed above, yields

$$PR(p_i;t+1) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j;t)}{L(p_j)}$$

where d is the damping factor,

or in matrix notation

$$\mathbf{R}(t+1) = d\mathcal{M}\mathbf{R}(t) + \frac{1-d}{N}\mathbf{1}, \qquad (1)$$

where $\mathbf{R}_i(t) = PR(p_i;t)$ and $\mathbf{1}$ is the column vector of length N containing only ones.

The matrix $\mathcal{M}$ is defined as

$$\mathcal{M}_{ij} = \begin{cases} 1/L(p_j), & \text{if } j \text{ links to } i \\ 0, & \text{otherwise} \end{cases}$$

i.e.,

$$\mathcal{M} := (K^{-1}A)^T,$$

where A denotes the adjacency matrix of the graph and K is the diagonal matrix with the outdegrees in the diagonal.

The probability calculation is made for each page at a time point, then repeated for the next time point. The computation ends when for some small ϵ

$$|\mathbf{R}(t+1) - \mathbf{R}(t)| < \epsilon,$$

i.e., when convergence is assumed.

演算法實作

```
def self_PageRank(input_data,arg):
    '''
    PageRank 實作
    '''
    start_time=time.time()
    #前處理###################################
    graph_1=to_graph(input_data)
    A_M=Adjacent_matrix(graph_1)
    ##########################################

    G_length=graph_1.vertex_num()#取得 N
    d=arg.damp#取得 damp d

    temp=np.array([np.sum(A_M,axis=1)]).T #取得每一節點的出度
    temp[temp==0]=1#不能造成除以零的狀況 (出度為零的話那一格 row 也都是零，所以維持一樣就好
    M_M=A_M/temp#做出 row normalized 鄰接矩陣 (每條邊標示成 1/父節點出度)
    pr=np.ones([G_length])*1/G_length#初始化 pagerank 為 1/N
    temp=np.matmul(M_M.T,pr)
    for itr in range(arg.itr):
        temp=np.matmul(M_M.T,pr)
        #參照老師 PPT 版本定義改寫 wiki 公式為 PR(t+1)=(1-d)MPR(t)+d/N, 其中，如果 j 點連
        pr=(d)/G_length+(1-d)*temp
    end_time=time.time()
    if arg.print_result:#如果要印出 result 可以將  --print_result 設為 True
        print("PageRank:",pr)
    if arg.show_time:#如果要印出 計算時間 可以將  --show_time 設為 True
        print("PageRank_Computation_time:",end_time-start_time)

    return pr
```

SimRank

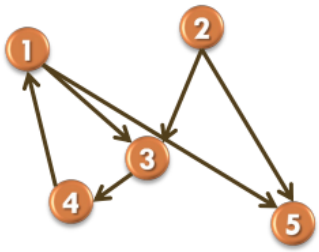
演算法參照 (老師講義第 52 頁, WIKI) :

SimRank

- SimRank formula

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

  - $I(a)$, $I(b)$: all in-neighbors
  - C is decay factot, 0<C<1
  - $S(a, b)\in[0, 1]$
  - $S(a, a)=1$



1'st iteration
S(3, 5)=C/4 * 2
S(4, 5)=0

How about S(4,5) while e(1,2) is added?

Data Mining

Matrix representation of SimRank [edit]

Given an arbitrary constant C between 0 and 1, let $\mathbf{S}$ be the similarity matrix whose entry $[\mathbf{S}]_{a,b}$ denotes the similarity score $s(a,b)$, and $\mathbf{A}$ be the column normalized adjacency matrix whose entry $[\mathbf{A}]_{a,b} = \frac{1}{|\mathcal{I}(b)|}$ if there is an edge from a to b, and 0 otherwise. Then, in matrix notations, SimRank can be formulated as

$$\mathbf{S} = \max\{C \cdot (\mathbf{A}^T \cdot \mathbf{S} \cdot \mathbf{A}), \mathbf{I}\},$$

where $\mathbf{I}$ is an identity matrix.

演算法實作

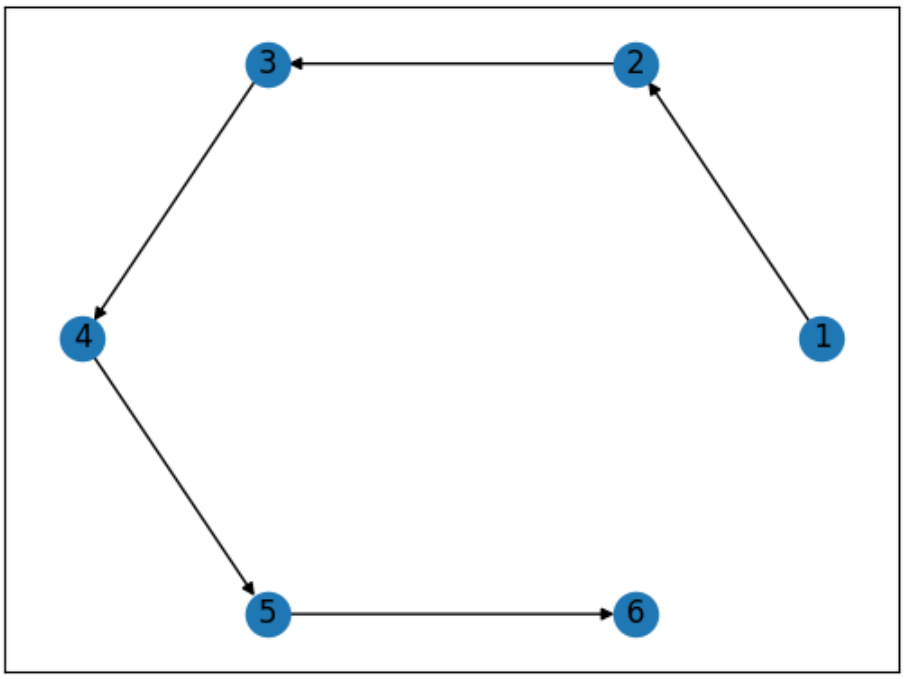
```
1
2  def max_matrix(a_M,b_M):
3      '''
4      設立 constraint: S(a,b)在 [0,1]，S(a,a)=1
5      '''
6      temp_M=np.zeros(a_M.shape)
7      for i in range(a_M.shape[0]):
8          for j in range(b_M.shape[1]):
9              if a_M[i][j]>b_M[i][j]:
10                 temp_M[i][j]=a_M[i][j]
11             else:
12                 temp_M[i][j]=b_M[i][j]
13     return temp_M
14
15 def self_SimRank(input_data,arg):
16     '''
17     SimRank 演算法的實作
18     '''
19     start_time=time.time()
20     #前處理####################################
21     graph_1=to_graph(input_data)
22     A_M=Adjacent_matrix(graph_1)
23     ##########################################
24
25     G_length=graph_1.vertex_num()#取得 N
26     c=arg.decay#取得 decay c
27     temp=np.array([np.sum(A_M,axis=0)])#取得各點入度
28     temp[temp==0]=1#不能造成除以零的狀況 (出度為零的話那一格 column 也都是零，所以維持一樣就
29     M_M=A_M/temp# column normalized 矩陣 (每條邊標示成 1/子節點入度)
30     Id=np.identity(G_length)# 單位矩陣
31     sim=np.identity(G_length)# 以單位矩陣初始化 SimRank
32     for itr in range(arg.itr):
33         #M_M.T dot Sim dot M_M 就會是 S(I(a),I(b))/|I(a)||I(b)|  之後乘上 decay c  就能
34           sim=max_matrix(np.dot(c,np.dot(np.dot(M_M.T,sim),M_M)),Id)
35     end_time=time.time()
36     if arg.print_result:#如果要印出 result 可以將  --print_result 設為 True
37         print("SimRank:")
38         print(sim)
39     if arg.show_time:#如果要印出 計算時間 可以將  --show_time 設為 True
40         print("SimRank_Computation_time:",end_time-start_time)
41     return sim
```

Result analysis and discussion

Graph_1

```
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: [0.017 0.032 0.045 0.057 0.068 0.078]
SimRank:
[[1. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 1.]]
```

graph_1 結果



graph_1

結果討論:

- 因為 node 1 沒有父節點，所以 authority 為 0
- 因為 node 6 沒有子節點，所以 hub 為 0
- PageRank 因為是單方向的傳遞所以越後面的點，PR值越高 (因為 node 1 沒有父節點, node 6 沒有子節點，PageRank 在計算上有瑕疵需藉由調整 damp 來令 PageRank 較能代表停留在頁面的機率)
- SimRank 從拓樸結構上沒有一個點與另一個點相似 (輸出是單位矩陣)，每一個點的位置都是獨一無二

更改 parameter

- 調高 damping (0.5): `PageRank: [0.083 0.125 0.146 0.156 0.161 0.164]`
  整體皆變高，因為隨機點選的比例佔高了

- 調低 damping(0.01): `PageRank: [0.002 0.003 0.005 0.007 0.008 0.01 ]`
  整體皆變低，因為隨機點選的比例佔低了

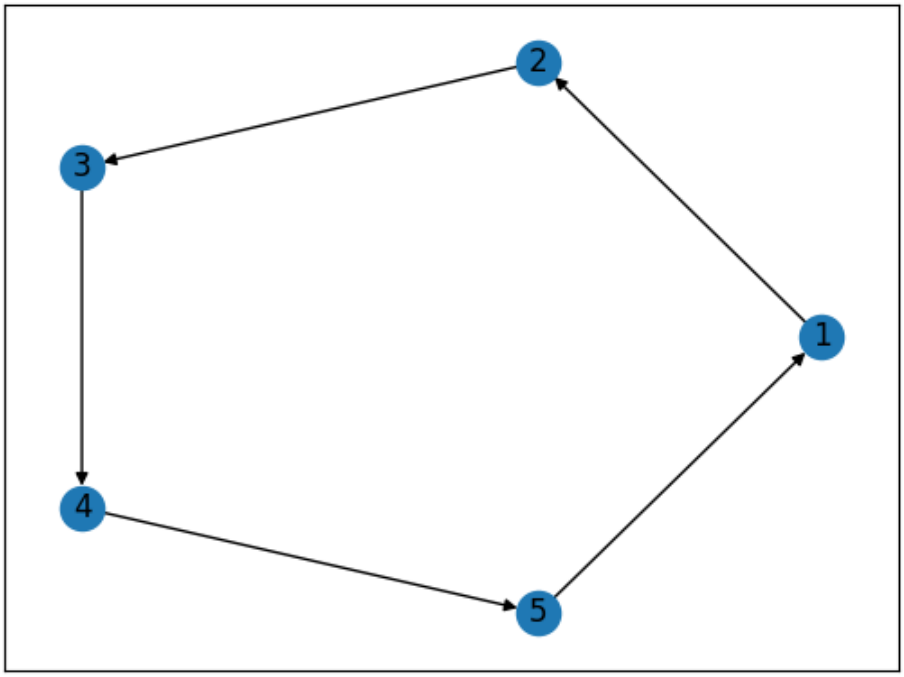
- 調高 decay(0.9):
```
SimRank:
[[1. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 1.]]
```
  沒有改變(因為本身相似度就低)

- 調低 decay(0.1):
```
SimRank:
[[1. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 1.]]
```
  沒有改變(因為本身相似度就低)

Graph_2

```
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]
SimRank:
[[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]]
```

graph_2 結果



graph_2

結果討論:

- 因為是一個環，所以 authority 皆相同為 1/N
- 因為是一個環，所以 hub 皆相同為 1/N
- 因為是一個環，所以 PageRank 皆相同為 1/N
- SimRank 從拓樸結構上沒有一個點與另一個點相似 (輸出是單位矩陣)，每一個點的位置都是獨一無二

更改 parameter

- 調高 damping (0.5): `PageRank: [0.2 0.2 0.2 0.2 0.2]`
  沒有改變 (因為是環，收束後大家都一樣是 1/N)
- 調低 damping(0.01): `PageRank: [0.2 0.2 0.2 0.2 0.2]`
  沒有改變 (因為是環，收束後大家都一樣是 1/N)

- 調高 decay(0.9):

```
SimRank:
[[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]]
```

  沒有改變(因為本身相似度就低)

- 調低 decay(0.1):

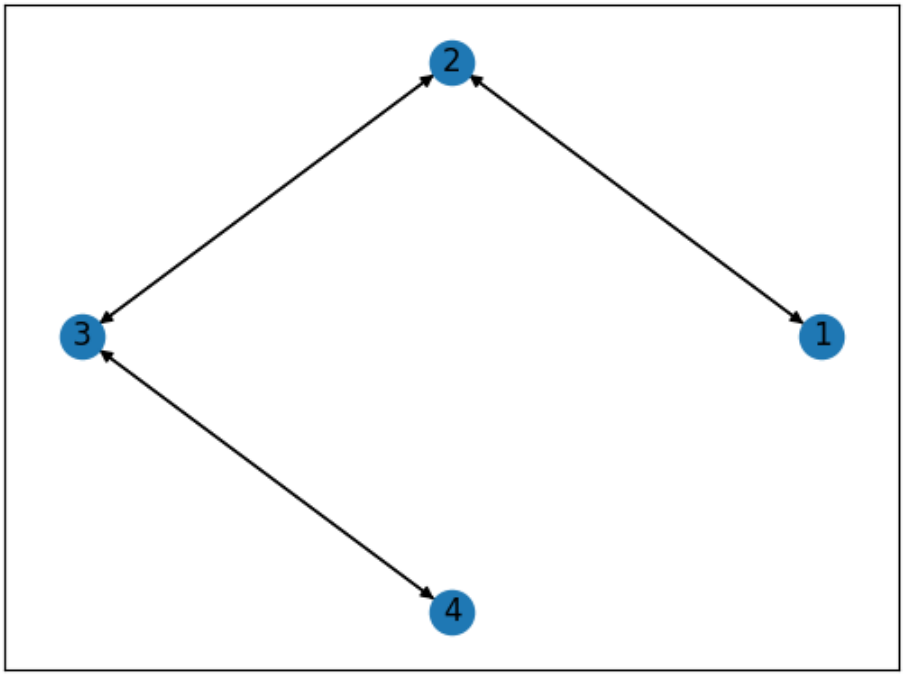
```
SimRank:
[[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]]
```

  沒有改變(因為本身相似度就低)

Graph_3

```
Authority: [0.191 0.309 0.309 0.191]
Hub: [0.191 0.309 0.309 0.191]
PageRank: [0.172 0.328 0.328 0.172]
SimRank:
[[1.    0.    0.538 0.   ]
 [0.    1.    0.    0.538]
 [0.538 0.    1.    0.   ]
 [0.    0.538 0.    1.   ]]
```

graph_3 結果



graph_3

結果討論:

- node 1,node 4 入度較少 因此 authority 比中間兩點低

- node 1,node 4 出度較少 因此 hub 比中間兩點低
- 若是隨機點取得化，中間兩點因為聯入的 link 較多，所以停止的機率比較大
- SimRank 從拓樸結構上 (1,3), (2,4) 有較大相似度

更改 parameter

- 調高 damping (0.5):`PageRank: [0.2 0.3 0.3 0.2]`
  node 1,4 變多， node 2,4 變小，全部皆與平均值 (1/N=0.25) 越來越靠近
- 再調高 damping (0.9):`PageRank: [0.238 0.262 0.262 0.238]`
  全部皆與平均值 (1/N=0.25) 極度靠近
- 調低 damping(0.01): `PageRank: [0.167 0.333 0.333 0.167]`
  node 1,4 ， node 2,4 ，兩組差距變的更明顯
- 調高 decay(0.9):
```
SimRank:
[[1.    0.    0.818 0.   ]
 [0.    1.    0.    0.818]
 [0.818 0.    1.    0.   ]
 [0.    0.818 0.    1.   ]]
```
  數值變高,decay 變高的話，每一次 iteration 傳下去的值就會變高
- 調低 decay(0.1):
```
SimRank:
[[1.    0.    0.053 0.   ]
 [0.    1.    0.    0.053]
 [0.053 0.    1.    0.   ]
 [0.    0.053 0.    1.   ]]
```
  數值變高,decay 變低的話，每一次 iteration 傳下去的值就會變低

Computation performance analysis

	HITS	PageRank	SimRank
graph_1	0.00009	0.00000	0.00199
graph_2	0.00000	0.00100	0.00099
graph_3	0.00100	0.0000	0.00100
graph_4	0.00099	0.00000	0.00299
graph_5	0.026	0.016	10.480
graph_6	0.048	0.030	76.675
ibm-5000	0.035	0.022	34.59

從計算複雜度來看，HITS 和 PageRank 量級相當（皆在一次 iteration 做一次矩陣運算）而 SimRank 要做兩次，且每一次皆須跑過全部得元素做 constraint 的判斷，因此時間最久。

Discussion

這一次實作中，將 lin analysis 的一些基礎演算法都實作了一遍。但也因為是基礎的演算法，可以透過一點投機的方法就能改善算出來的分數。像是指向自己，就可以對 Authority, Hub, PageRank 有很大的影響。所以後續也陸續有改良的演算法法推出，防止投機的情形。在改變 parameter 時也觀察

到，使用者的操作是無法預測的，及使用個一個較廣泛的參數，也很難說這就是實際上網頁 link 的結果，只能多方嘗試產生 data 然後繼續分析。