Information Retrieval and Text Mining: Homework #4

R08725008 資管碩二 周若涓

執行環境

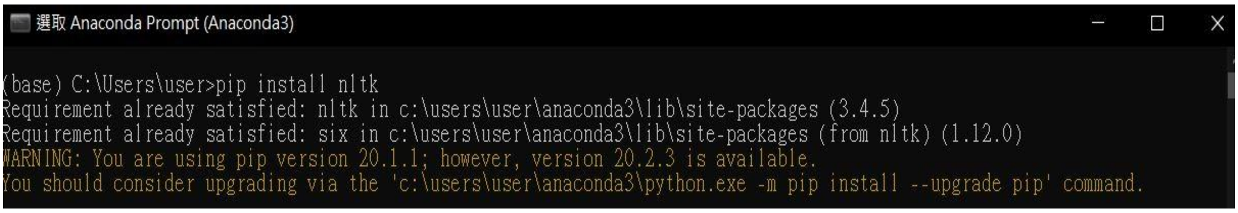
- Jupyter Notebook

程式語言

- Python 3

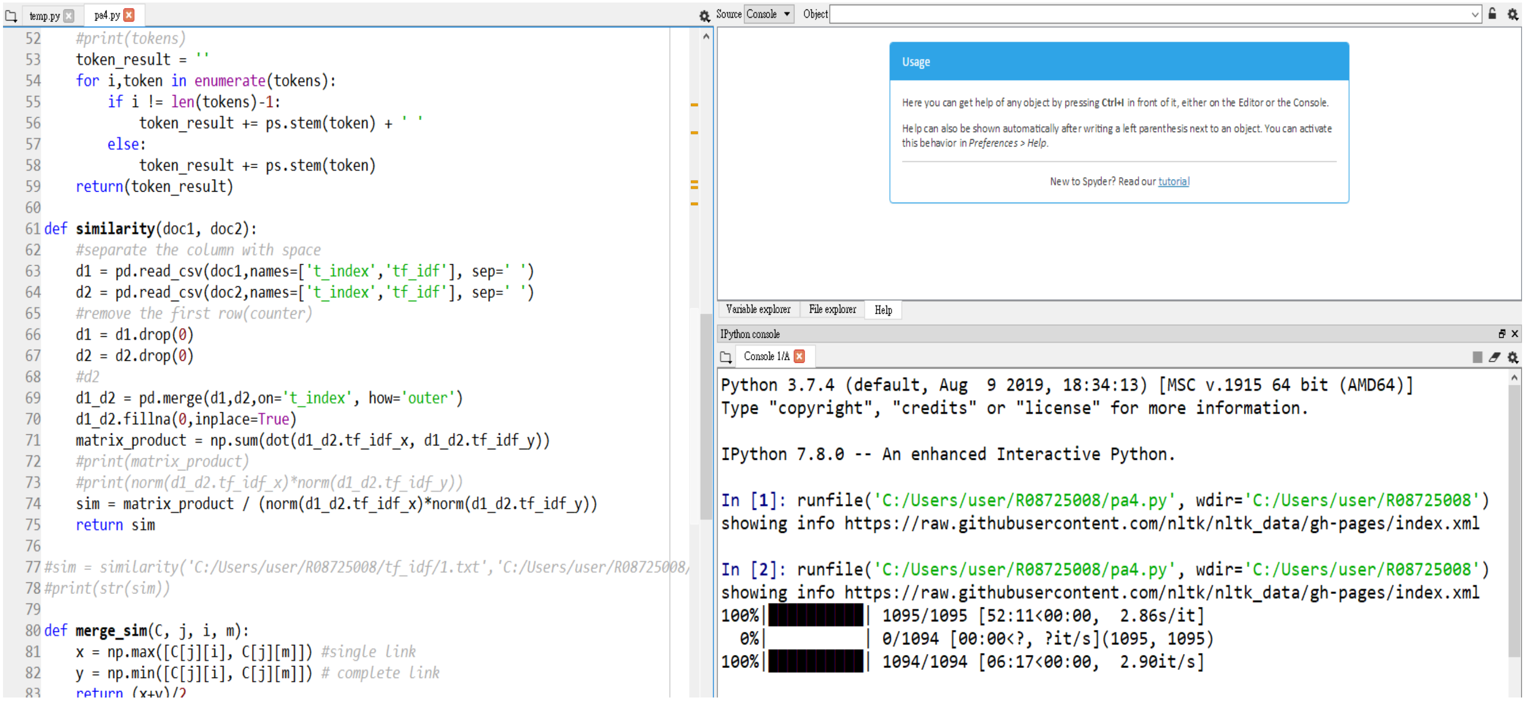
執行方式

- 在執行前需安裝 nltk 套件 (command: pip install nltk)



```
(base) C:\Users\user>pip install nltk
Requirement already satisfied: nltk in c:\users\user\anaconda3\lib\site-packages (3.4.5)
Requirement already satisfied: six in c:\users\user\anaconda3\lib\site-packages (from nltk) (1.12.0)
WARNING: You are using pip version 20.1.1; however, version 20.2.3 is available.
You should consider upgrading via the 'c:\users\user\anaconda3\python.exe -m pip install --upgrade pip' command.
```

- 安裝 pandas 套件 (command: pip install pandas)
- 安裝 numpy 套件 (command: pip install numpy)
- 執行環境:
    - 可以利用 Spyder 開啟 pa4.py，並執行



```python
    #print(tokens)
    token_result = ''
    for i,token in enumerate(tokens):
        if i != len(tokens)-1:
            token_result += ps.stem(token) + ' '
        else:
            token_result += ps.stem(token)
    return(token_result)

def similarity(doc1, doc2):
    #separate the column with space
    d1 = pd.read_csv(doc1,names=['t_index','tf_idf'], sep=' ')
    d2 = pd.read_csv(doc2,names=['t_index','tf_idf'], sep=' ')
    #remove the first row(counter)
    d1 = d1.drop(0)
    d2 = d2.drop(0)
    #d2
    d1_d2 = pd.merge(d1,d2,on='t_index', how='outer')
    d1_d2.fillna(0,inplace=True)
    matrix_product = np.sum(dot(d1_d2.tf_idf_x, d1_d2.tf_idf_y))
    #print(matrix_product)
    #print(norm(d1_d2.tf_idf_x)*norm(d1_d2.tf_idf_y))
    sim = matrix_product / (norm(d1_d2.tf_idf_x)*norm(d1_d2.tf_idf_y))
    return sim

#sim = similarity('C:/Users/user/R08725008/tf_idf/1.txt','C:/Users/user/R08725008/
#print(str(sim))

def merge_sim(C, j, i, m):
    x = np.max([C[j][i], C[j][m]]) #single link
    y = np.min([C[j][i], C[j][m]]) # complete link
    return (x+y)/2
```

Usage

Here you can get help of any object by pressing Ctrl+I in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our tutorial

```
Python 3.7.4 (default, Aug  9 2019, 18:34:13) [MSC v.1915 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.8.0 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/user/R08725008/pa4.py', wdir='C:/Users/user/R08725008')
showing info https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml

In [2]: runfile('C:/Users/user/R08725008/pa4.py', wdir='C:/Users/user/R08725008')
showing info https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml
100%|██████████| 1095/1095 [52:11<00:00,  2.86s/it]
  0%|          | 0/1094 [00:00<?, ?it/s](1095, 1095)
100%|██████████| 1094/1094 [06:17<00:00,  2.90it/s]
```

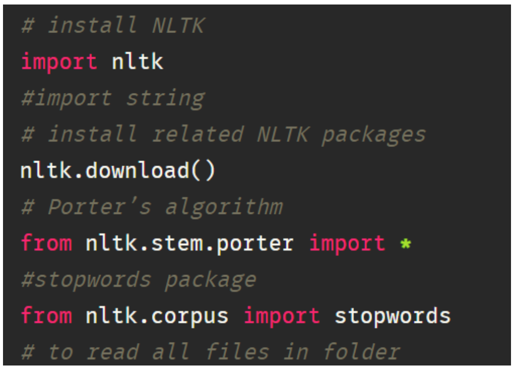
- 確保提供的 IRTM 資料夾放於 C:/Users/user/R08725008/ 目錄下
- 確保 tf_idf 資料夾放於 C:/Users/user/R08725008/ 目錄下
- 確保中間產出的 C.pkl 放於 C:/Users/user/R08725008/ 目錄下
- 產出的 8.txt, 13.txt, 20.txt 預設放於 C:/Users/user/R08725008/ 目錄下

作業邏輯說明

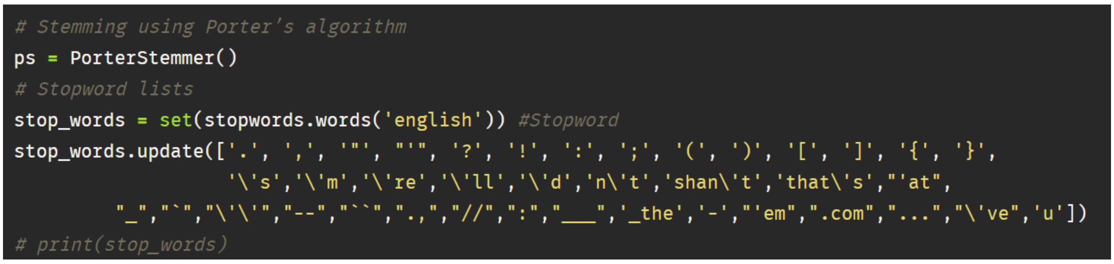
Part 1: Preprocessing

1.

- 利用 nltk 套件初始化 PorterStemmer
- import stopwords 套件，並增加 list，其為觀察到可能為 stopwords



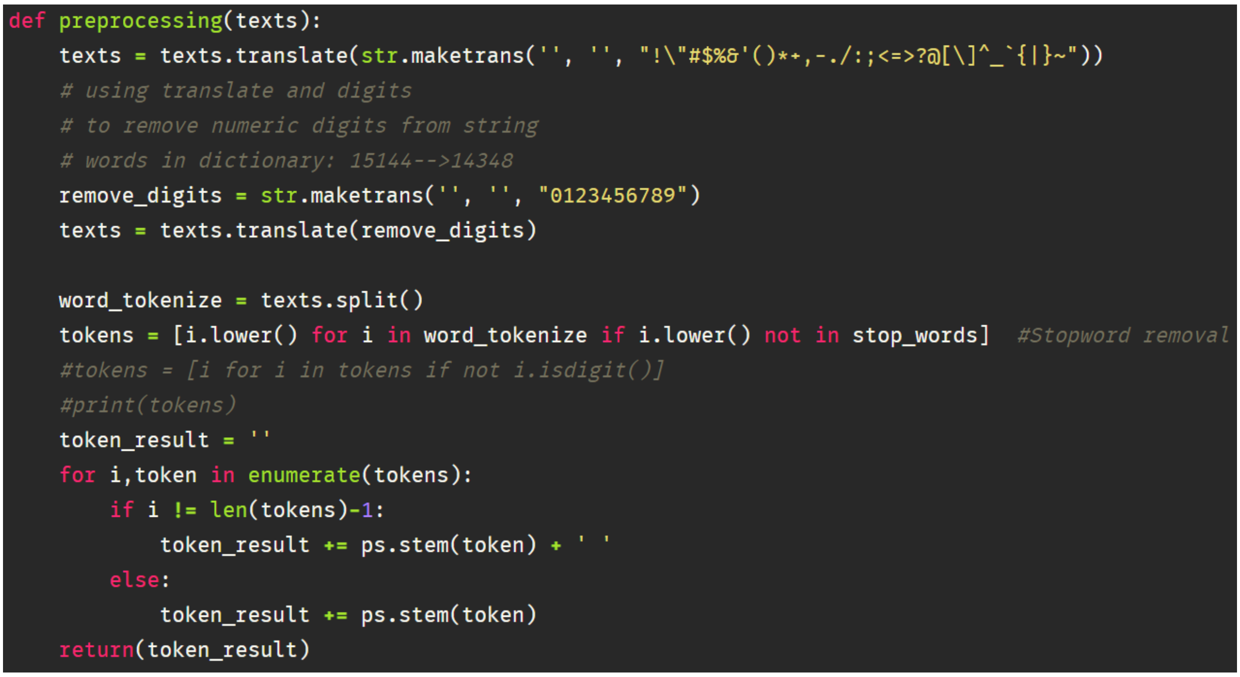
```
# install NLTK
import nltk
#import string
# install related NLTK packages
nltk.download()
# Porter’s algorithm
from nltk.stem.porter import *
#stopwords package
from nltk.corpus import stopwords
# to read all files in folder
```



```
# Stemming using Porter’s algorithm
ps = PorterStemmer()
# Stopword lists
stop_words = set(stopwords.words('english')) #Stopword
stop_words.update(['.', ',', '"', "'", '?', '!', ':', ';', '(', ')', '[', ']', '{', '}',
                   '\'s','\'m','\'re','\'ll','\'d','n\'t','shan\'t','that\'s',"'at",
         "_","`","\'\'","--","``",".,","//",":","___",'_the','-',"'em",".com","...","\'ve",'u'])
# print(stop_words)
```

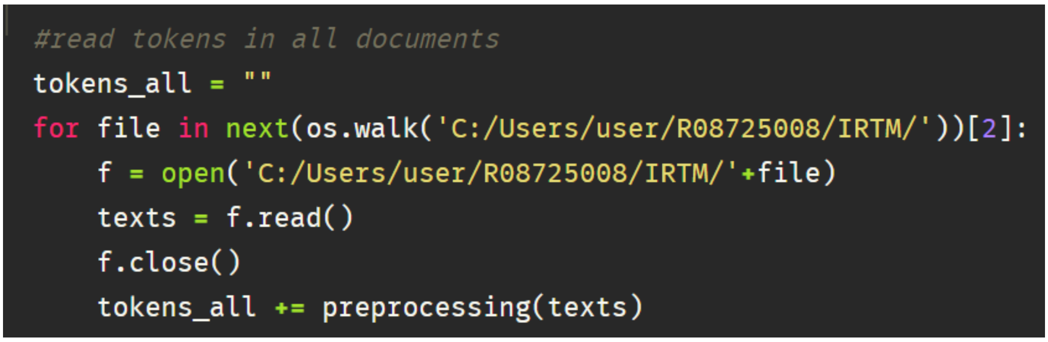
2. 定義 preprocessing function，有以下功能

- 去除 punctuation
- 加上去除數字的處理
- Tokenize
- 將讀入的文件轉換為小寫，如果不在 stop words 當中的 word 才會保留
- 最後將每個 token 進行 stemming，並加回字串當中
- 把結果輸出 token_result



```python
def preprocessing(texts):
    texts = texts.translate(str.maketrans('', '', "!\"#$%&'()*+,-./:;<=>?@[\]^_`{|}~"))
    # using translate and digits
    # to remove numeric digits from string
    # words in dictionary: 15144-->14348
    remove_digits = str.maketrans('', '', "0123456789")
    texts = texts.translate(remove_digits)

    word_tokenize = texts.split()
    tokens = [i.lower() for i in word_tokenize if i.lower() not in stop_words]  #Stopword removal
    #tokens = [i for i in tokens if not i.isdigit()]
    #print(tokens)
    token_result = ''
    for i,token in enumerate(tokens):
        if i != len(tokens)-1:
            token_result += ps.stem(token) + ' '
        else:
            token_result += ps.stem(token)
    return(token_result)
```

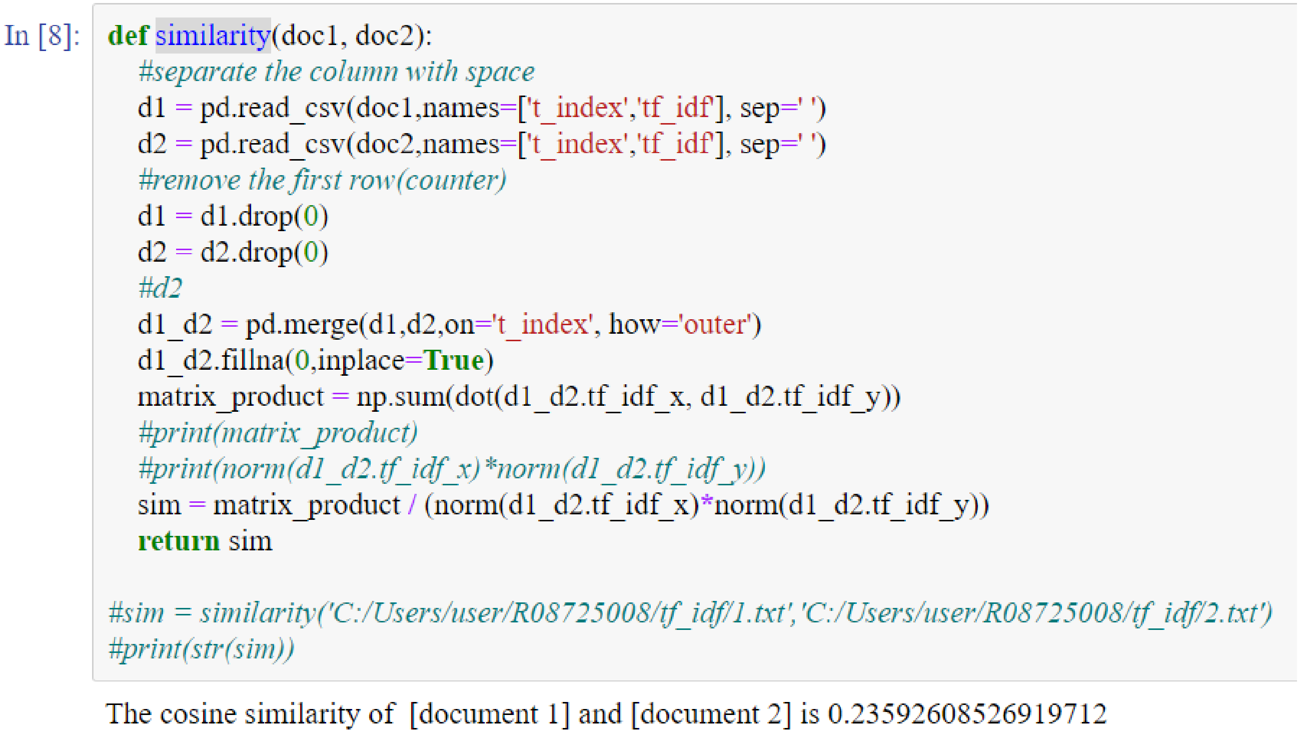
3. 讀入所有資料夾中的 documents，經過前處理後的所有 terms 收集至 tokens_all



```python
#read tokens in all documents
tokens_all = ""
for file in next(os.walk('C:/Users/user/R08725008/IRTM/'))[2]:
    f = open('C:/Users/user/R08725008/IRTM/'+file)
    texts = f.read()
    f.close()
    tokens_all += preprocessing(texts)
```

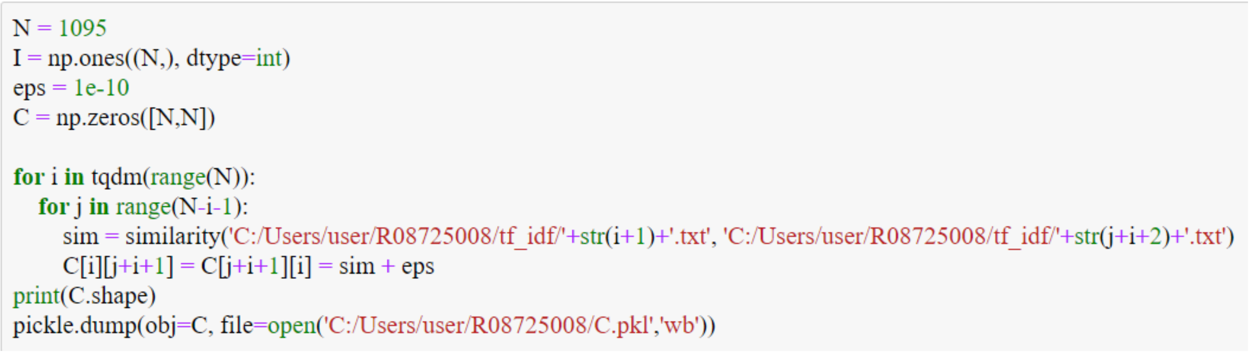
Part 2: Similarity Matrix C

1. 定義 similarity function 計算 cosine similarity
    - 讀取 document 1 以及 document 2
    - 去除第一列 (此 document 中在 token_list 中的 terms 的數量)
    - cosine similarity 分子為向量相乘，使用 numpy 中的 dot 運算
    - cosine similarity 分母為向量長度相乘，使用 numpy 中的 norm 運算來計算向量的長度
    - 上述兩點相除，即得到 cosine similarity，此為 function 的 return value



```
In [8]:  def similarity(doc1, doc2):
             #separate the column with space
             d1 = pd.read_csv(doc1,names=['t_index','tf_idf'], sep=' ')
             d2 = pd.read_csv(doc2,names=['t_index','tf_idf'], sep=' ')
             #remove the first row(counter)
             d1 = d1.drop(0)
             d2 = d2.drop(0)
             #d2
             d1_d2 = pd.merge(d1,d2,on='t_index', how='outer')
             d1_d2.fillna(0,inplace=True)
             matrix_product = np.sum(dot(d1_d2.tf_idf_x, d1_d2.tf_idf_y))
             #print(matrix_product)
             #print(norm(d1_d2.tf_idf_x)*norm(d1_d2.tf_idf_y))
             sim = matrix_product / (norm(d1_d2.tf_idf_x)*norm(d1_d2.tf_idf_y))
             return sim

         #sim = similarity('C:/Users/user/R08725008/tf_idf/1.txt','C:/Users/user/R08725008/tf_idf/2.txt')
         #print(str(sim))

         The cosine similarity of  [document 1] and [document 2] is 0.23592608526919712
```

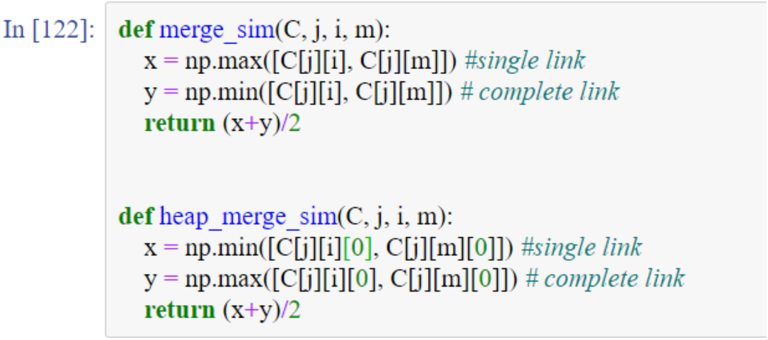
2. 將 document 做 pairwise 的 similarity 計算，以防之後會產生 zero probability 的問題，因此每個 matrix 的 entry 皆加上一個極小的數值
    - 並把計算結果的 C matrix dump 出來



```
N = 1095
I = np.ones((N,), dtype=int)
eps = 1e-10
C = np.zeros([N,N])

for i in tqdm(range(N)):
    for j in range(N-i-1):
        sim = similarity('C:/Users/user/R08725008/tf_idf/'+str(i+1)+'.txt', 'C:/Users/user/R08725008/tf_idf/'+str(j+i+2)+'.txt')
        C[i][j+i+1] = C[j+i+1][i] = sim + eps
print(C.shape)
pickle.dump(obj=C, file=open('C:/Users/user/R08725008/C.pkl','wb'))
```

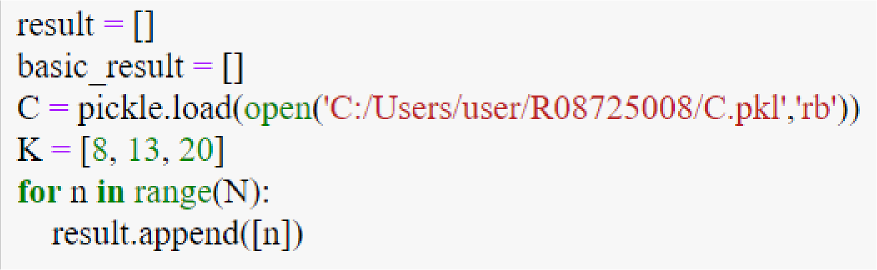
Part 3: Implementing HEAP Efficiency HAC

1. 利用 single link 與 complete link 的值取平均計算 cluster 之間的 similarity



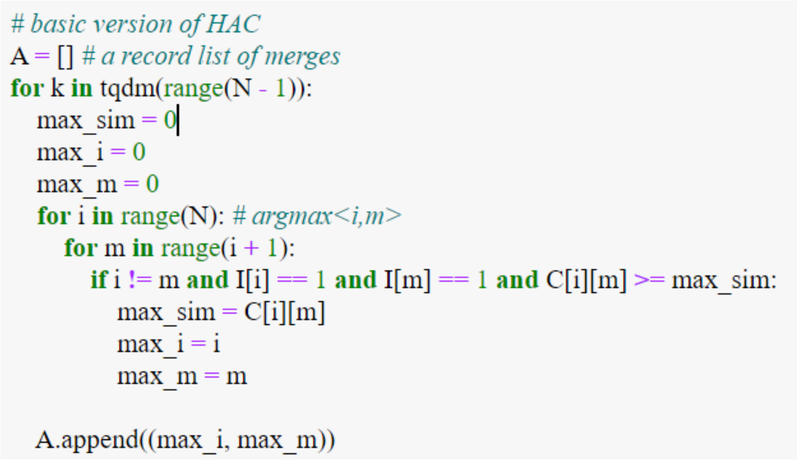
```
In [122]:  def merge_sim(C, j, i, m):
               x = np.max([C[j][i], C[j][m]]) #single link
               y = np.min([C[j][i], C[j][m]]) # complete link
               return (x+y)/2


           def heap_merge_sim(C, j, i, m):
               x = np.min([C[j][i][0], C[j][m][0]]) #single link
               y = np.max([C[j][i][0], C[j][m][0]]) # complete link
               return (x+y)/2
```

2. 初始化 result，一開始還沒進行合併時，每個 document 各自形成一個 cluster，並且把上一步驟所得到 C matrix load 出來使用



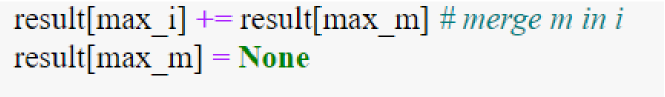
```
result = []
basic_result = []
C = pickle.load(open('C:/Users/user/R08725008/C.pkl','rb'))
K = [8, 13, 20]
for n in range(N):
    result.append([n])
```

3. 初始化 A，A 是用來記錄 merge 的過程，哪個 cluster 與哪個 cluster 做合併
    - 因為合併過程有 N-1 個 internal nodes，所以要做 N-1 次 merge
    - 找到最大的 C[i][m] 值
    - 找到後，再把 (i,m) append 至 A list



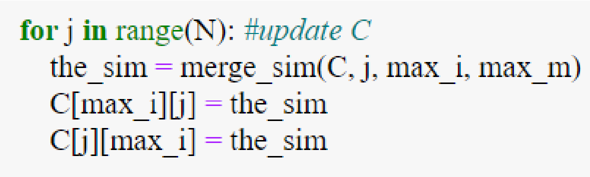
```
# basic version of HAC
A = [] # a record list of merges
for k in tqdm(range(N - 1)):
    max_sim = 0
    max_i = 0
    max_m = 0
    for i in range(N): # argmax<i,m>
        for m in range(i + 1):
            if i != m and I[i] == 1 and I[m] == 1 and C[i][m] >= max_sim:
                max_sim = C[i][m]
                max_i = i
                max_m = m

    A.append((max_i, max_m))
```

4. 把 m cluster 合併到 i cluster，m cluster 設為 None



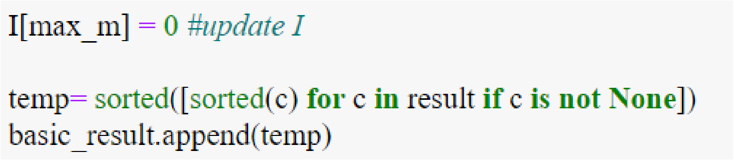
```
result[max_i] += result[max_m] # merge m in i
result[max_m] = None
```

5. 更新其他 cluster 與 cluster i 之間的 similarity



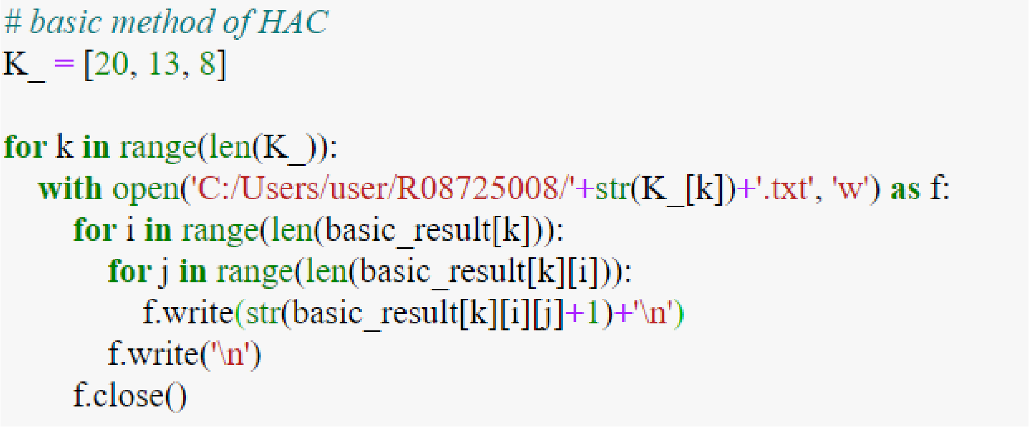
```
for j in range(N): #update C
    the_sim = merge_sim(C, j, max_i, max_m)
    C[max_i][j] = the_sim
    C[j][max_i] = the_sim
```

6. Update I，因為 m cluster 被合併了，再將 document 以 id 大小做排序



```
I[max_m] = 0 #update I

temp= sorted([sorted(c) for c in result if c is not None])
basic_result.append(temp)
```

7. 把分群結果 output 出來



```
# basic method of HAC
K_ = [20, 13, 8]

for k in range(len(K_)):
    with open('C:/Users/user/R08725008/'+str(K_[k])+'.txt', 'w') as f:
        for i in range(len(basic_result[k])):
            for j in range(len(basic_result[k][i])):
                f.write(str(basic_result[k][i][j]+1)+'\n')
            f.write('\n')
        f.close()
```