1. 執行環境: VS code
2. 程式語言: Python 3.11.5
3. 執行需求:
   - 文章資料放置在./data/ 路徑之下
   - 8.txt,13.txt,20.txt 最終結果會放置與 pa3.py 同一層資料夾
   - 需要 pip install numpy, nltk, pandas
4. 作業處理邏輯

Step1. 將資料讀進，並各自做前處理(preprocessing)後算出 tf 與 df，分別存進 word_tf 與 word_df 變數中

```
def preprocessing(file):
    file = file.lower()
    file = Tokenize(file)
    file = RemoveStopwords(file,StopWords)
    file = Porter_Stemmmer(file)
    return file

def TermFrequency(start,end):
    word_tf =[]
    word_df = {}
    for i in range(start,end+1):
        i=str(i)
        print(i)
        path = f'./data/{i}.txt'
        tdf = OpenFile(path)
        tdf = preprocessing(tdf)
        tdf = [word for word in tdf if word!='']
        for word in tdf:
            if word not in word_tf:
                temp =[]
                frequency = tdf.count(word)
                temp.append(i)
                temp.append(word)
                temp.append(frequency)
                word_tf.append(temp)
        for word in set(tdf):
            if word in word_df:
                word_df[word] +=1
            else :
                word_df[word] =1

    return word_tf, word_df
word_tf, word_df = TermFrequency(1,size)
```

Step2.　建立字典並編號，存在 index_dict 變數裡

```
def get_index_dict(word_df):
    index_dict={}
    index =0
    for term in word_df:
        index_dict[term] = index
        index +=1
    return index_dict

index_dict = get_index_dict(word_df)
```

Step3.　將文件轉換成 tf-idf vector，並撰寫可以計算 cosine similarity 的公式

```
def get_tf_vector(word_tf,index_dict):
    tf_vectors=[]
    for i in range(1,size+1): #
        tf_vector = np.array([0] * len(index_dict), dtype=float)
        for row in word_tf:
            if row[0] == str(i):
                word = row[1]
                tf_vector[index_dict[word]] = row[2]
            else:
                continue
        tf_vectors.append([i, tf_vector])
    return tf_vectors

tf_vectors = get_tf_vector(word_tf,index_dict)
import math
def get_tf_idf_vector(tf_vectors, word_df, index_dict):
    idf_vector = np.array([0] * len(index_dict), dtype=float)
    for word, df in word_df.items():
        idf = math.log(len(tf_vectors) / df, size)
        idf_vector[index_dict[word]] = idf
    tf_idf_vectors = list()
    for tf_vector in tf_vectors:
        idx = tf_vector[0]
        print(tf_vector[1])
        tf_idf = np.array(tf_vector[1], dtype=float) *
np.array(idf_vector, dtype=float)
        tf_idf_unit = tf_idf / np.linalg.norm(tf_idf)
        tf_idf_vectors.append([idx, tf_idf_unit])
    return tf_idf_vectors

tf_idf_vectors = get_tf_idf_vector(tf_vectors,word_df,index_dict)

doc_vectors = np.array([each[1] for each in tf_idf_vectors])
def extract_vector(doc_id, doc_vectors):
    return doc_vectors[doc_id-1]

def cosine(doc_x, doc_y):
    vector_x = extract_vector(doc_x, doc_vectors)
    vector_y = extract_vector(doc_y, doc_vectors)
    cosine_sim = float(np.dot(vector_x, vector_y))
    return cosine_sim
te = cosine(1,2)
```

Step4.　使用 complete-link 策略並按照 HAC 的 algothrim 撰寫其餘程式碼

```
def get_max_similarity(C, I, size):
    max_sim = -1
    doc_i, doc_m = -1, -1
    for i in range(size):
        if I[i] != 1:
            continue
        for m in range(size):
            if I[m] == 1 and i != m:
                if max_sim < C[i][m]:
                    max_sim = C[i][m]
                    doc_i, doc_m = i, m
    return doc_i, doc_m
```

```
C = np.array([[cosine(x, y) for x in range(1, size+1)] for y in
range(1, size+1)])
I = np.array([1]* size)
A = []
for k in range(1,size-1):
    i, m = get_max_similarity(C, I, size)
    A.append([i ,m])
    for j in range(size):
        C[i][j] = min(cosine(i, j), cosine(m, j))
        C[j][i] = min(cosine(j, i), cosine(j, m))
    I[m] = 0
```

Step5.　保存結果，分別存入 8.txt,13.txt,20.txt

```
def write_result(hac_dict, cluster_num):
    with open(f"./{cluster_num}.txt", "w") as f:
        for k, v in hac_dict.items():
            for doc_id in sorted(v):
                f.write(f"{doc_id+1}\n")
            f.write("\n")

hac_dict = {str(i) : [i] for i in range(size)}

for doc_i, doc_m in A:
    new_element = hac_dict[str(doc_m)]
    hac_dict.pop(str(doc_m))
    hac_dict[str(doc_i)] += new_element
    if len(hac_dict) == 20:
        write_result(hac_dict, 20)
    if len(hac_dict) == 13:
        write_result(hac_dict, 13)
    if len(hac_dict) == 8:
        write_result(hac_dict, 8)
```