文字探勘 HW3
資管三 B11705033 江睿宸
1. 執行環境：Jupyter Notebook
2. 程式語言：Python3
3. 程式碼說明

匯入需要的套件

In [1]:
```
import os
import glob
import nltk
from nltk.stem import PorterStemmer
import numpy as np
from collections import defaultdict
import math
import heapq
import copy

nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\sandy\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Out[1]: True

使用作業二計算出的 vector (位於 output 中)計算 cosine similarity

In [3]:
```
# 讀取檔案
def load_vector(doc_name):
    vector = {}
    with open(doc_name, 'r', encoding='utf-8') as file:
        next(file)  # The number of terms document has
        for line in file:
            index, value = line.split()
            vector[int(index)] = float(value)
    return vector

def cosine(docx, docy):
    vector_x = load_vector(docx)
    vector_y = load_vector(docy)

    # 所有出現的單字集
    all_indices = set(vector_x.keys()).union(set(vector_y.keys()))

    # 統一長度將沒出現的單字填入0
    tf_idf_x = np.array([vector_x.get(term, 0) for term in all_indices])
    tf_idf_y = np.array([vector_y.get(term, 0) for term in all_indices])
    # Calculate the cosine similarity
    dot = np.dot(tf_idf_x, tf_idf_y)
    len_x = np.linalg.norm(tf_idf_x)
    len_y = np.linalg.norm(tf_idf_y)
    if len_x == 0 or len_y == 0:
        return 0.0

    cosine_similarity = dot / (len_x * len_y)
    return round(cosine_similarity, 10)
```

使用 merge_clusters 合併 cluster

使用 upadate_clusters 將合併前的 cluters 刪除後新增新的 cluster

In [5]:
```
def merge_clusters(cluster1, cluster2):
#     print("merge", cluster1, cluster2)
    return list(set(cluster1 + cluster2))
```

In [6]:
```
def update_clusters(clusters, cluster1_index, cluster2_index, merged_cluster):
    updated_clusters = [
        clusters[i] for i in range(len(clusters))
        if i not in [cluster1_index, cluster2_index]
    ]
    updated_clusters.append(merged_cluster)
    return updated_clusters
```

Calculate_distance 則重新計算 cluster 之間的距離，本題使用 single，使用 complete 會發現最後幾筆最大距離都是 1，也就是相似性 0，因此取距離最小

In [7]:
```
def calculate_distance(cluster1, cluster2):
    dis = min(distance_matrix[min(i, j), max(i, j)] for i in cluster1 for j in cluster2)
    return dis
```

實作 Heap 的功能

In [8]:
```
def push_to_heap(heap, value):
    """將一個元素插入到最小堆中，並保持堆的性質。"""
    heap.append(value)
    sift_up(heap, len(heap) - 1)

def pop_from_heap(heap):
    """移除並返回最小堆的堆頂元素。"""
    if not heap:
        return None
    heap[0], heap[-1] = heap[-1], heap[0]
    min_value = heap.pop()
    sift_down(heap, 0)
    return min_value

def sift_up(heap, index):
    """從指定索引位置向上調整，使最小堆性質成立。"""
    while index > 0:
        parent = (index - 1) // 2
        if heap[index][0] < heap[parent][0]:  # 比較相似度（最小值）
            heap[index], heap[parent] = heap[parent], heap[index]
            index = parent
        else:
            break
```

```
def sift_down(heap, index):
    """從指定索引位置向下調整，使最小堆性質成立。"""
    size = len(heap)
    while index < size:
        left_child = 2 * index + 1
        right_child = 2 * index + 2
        smallest = index

        if left_child < size and heap[left_child][0] < heap[smallest][0]:
            smallest = left_child
        if right_child < size and heap[right_child][0] < heap[smallest][0]:
            smallest = right_child

        if smallest != index:
            heap[index], heap[smallest] = heap[smallest], heap[index]
            index = smallest
        else:
            break
```

計算初始文件的相似度，並用 1-similarity 作為距離，為了避免 I/O 使用 distance_matrix 將一開始計算的 cosine_similarity 紀錄，不用反覆開檔案。

實作 min_heap 將結果存在 heap 中

In [9]:
```
distance_matrix = np.full((data_num, data_num), np.inf)
def calculate_initial_distances():
    heap = []
    for i in range(data_num):
        print(i)
        for j in range(i + 1, data_num):
            distance = cosine(f"output/{i+1}.txt", f"output/{j+1}.txt")
            push_to_heap(heap, (1 - distance, (i, j)))  # Push the tuple into the heap
            distance_matrix[i, j] = 1 - distance
    return heap
```

當 cluster 被合併刪除時會產生位移，因使用 index_map 確保合併過後編號還指向正確的位置，並根據結果重新計算距離

In [14]:
```
def HAC(K, heap, clusters):
    index_map = {i: i for i in range(len(clusters))}
    original_len = 1095
    while len(clusters) > K:
        distance, (original_cluster1_idx, original_cluster2_idx) = pop_from_heap(heap)
        # Map the original indices to the current indices
        cluster1_idx = index_map.get(original_cluster1_idx)
        cluster2_idx = index_map.get(original_cluster2_idx)

        if cluster1_idx is None or cluster2_idx is None:
            continue

        cluster1 = clusters[cluster1_idx]
        cluster2 = clusters[cluster2_idx]
        merged_cluster = merge_clusters(cluster1, cluster2)
        clusters = update_clusters(clusters, cluster1_idx, cluster2_idx, merged_cluster)

        # **Update index_map for all clusters**
        new_cluster_idx = len(clusters) - 1
        for key, val in index_map.items():
            if val > cluster1_idx:
                index_map[key] -= 1
            if val > cluster2_idx:
                index_map[key] -= 1
        # Remove old indices from the map
        index_map.pop(original_cluster1_idx, None)
        index_map.pop(original_cluster2_idx, None)

#         Calculate new distances and add them to the heap
        for key, val in index_map.items():

            new_distance = calculate_distance(clusters[val], merged_cluster)
            push_to_heap(heap, (1-new_distance, (key, original_len)))
        index_map[original_len] = len(clusters) - 1
        original_len += 1

    return clusters
```

執行並將結果存為指定檔案格式

In [11]:
```
heap_init = calculate_initial_distances()
```

In [16]:
```
K = 8
clusters = [[i] for i in range(data_num)]
heap = heap_init.copy()
clusters = HAC(K, heap, clusters)
save_clusters_to_file(clusters, f"{K}.txt")
```