1. 執行環境：Jupyter Notebook
2. 程式語言：Python 3
3. 執行方式：
    1. Open terminal
    2. $ python3 pa4.py
    4. Output (txt file): 8.txt, 13.txt, 20.txt

4. 作業處理邏輯說明：
    1. 載入套件

```
import numpy as np
```

    2. Function cosine (copy from pa2)：
    這次作業沿用 pa2 中計算 cosine similarity 的函式

```
def cosine(Docx, Docy):
    '''
    Input two document id to calculate their cosine similarity
    through tf-idf unit vector
    '''
    dict_x = {}  # dictionary of document x
    dict_y = {}  # dictionary of document y

    with open('./output/doc' + str(Docx) + '.txt',
              'r') as txt:  # read file write to dictionary
        content_x = [
            i.replace("\n", "").split("\t") for i in txt.readlines()[2:]
        ]

    for item in content_x:
        dict_x.update({item[0]: item[1]})

    with open('./output/doc' + str(Docy) + '.txt',
              'r') as txt:  # read file write to dictionary
        content_y = [
            i.replace("\n", "").split("\t") for i in txt.readlines()[2:]
        ]

    for item in content_y:
        dict_y.update({item[0]: item[1]})

    for item in dict_x:  # check items in another dict(y), if non-exist->give 0
        if item not in dict_y:
            dict_y.update({item: 0})

    for item in dict_y:  # check item in another dict(x), if non-exist->give 0
        if item not in dict_x:
            dict_x.update({item: 0})

    x = []  # create list x[] to store tf-idf score of document x
    y = []  # create list y[] to store tf-idf score of document y

    for key in sorted(dict_x):  # call sorted dictionary x
        x.append(float(dict_x.get(key)))  # get value: string to float

    for key in sorted(dict_y):  # call sorted dictionary y
        y.append(float(dict_y.get(key)))  # get value of y

    x = np.array(x)  # convert list x to array for calculating cosine similarity
    y = np.array(y)  # convert list y to array for calculating cosine similarity

    similarity_scores = np.dot(x,y) / np.sqrt(np.dot(x, x) * np.dot(y, y))

    return similarity_scores
```

3. 計算兩兩一對的 cosine similarity 並存至 matrix C

```
'''Calculate pair-wise similarity'''
col, row = 1096, 1096
C = [[0 for _ in range(row)] for _ in range(col)]  # create 1095*1095 C matrix
for i in range(1, 1096):
    for j in range(1, 1096):
        C[i][j] = cosine(i,j)  # calculate pair-wise similarity
    # if(i%10 == 0): print(i)
```

4. Max Heap:

為了加速分群，於這次作業中實作 Max Heap 以實現 Efficient priority-queue based HAC algorithm，用來在每次 iteration 中快速找到最大的 similarity。類別 Max_Heap 包含以下較重要的 function：

- push: 用來新增元素到 max heap，並做 bottom-up heapify
- pop: 用來取出 root element，並做 top-down heapify
- delete: 刪除 heap 中某一節點，並以刪除的該 index 出發做 top-down heapify
- peek: 用來查看 root element
- heapify: 可分成 top-down 跟 bottom-up，主要是確保 heap 在新增/修改/刪除等動作之後仍維持合法的 heap。

```
class Max_Heap:

    # initializing the constructor with array that we have to convert into heap, default value is None([])
    def __init__(self, arr=[]):
        # Initialization
        self._heap = []

        # If the array by the user is not empty, push all the elements
        if arr is not None:
            for root in arr:
                self.push(root)


    # push: insert new item to the heap and heapify
    def push(self, value):
        # appending the value given by user at the last
        self._heap.append(value)

        # use bottom_up() to ensure order of heap
        _bottom_up(self._heap, len(self) - 1)


    # pop: get root item and heapify
    def pop(self):
        if len(self._heap)!=0:
            # swap root with the last item
            _swap(self._heap, len(self) - 1, 0)

            # store the popped item to root
            root = self._heap.pop()

            # use _top_down() to ensure that the heap is still in legal
            _top_down(self._heap, 0)
        else:
            root = "Heap is empty"
        return root


    # delete: delete node by value
    def delete(self, value):
        if len(self._heap)!=0:
            del_idx = self._heap.index(value)

            # swap remove item value with the last item
            _swap(self._heap, len(self) - 1, del_idx)

            # remove item from heap list(針對 list item 本身 remove)
            self._heap.remove(value)

            # use _top_down() to ensure that the heap is still in legal
            _top_down(self._heap, del_idx)
```

```
    # print the first element (The root)
    def peek(self):
        if len(self._heap)!=0:
            return(self._heap[0])
        else:
            return("Heap is empty")


    # show the heap
    def display(self):
        return self._heap


    # Swaps i and j in heap
def _swap(L, i, j):
    L[i], L[j] = L[j], L[i]


    # private: heapify
def _bottom_up(heap, index):
    # Finding parent of the element
    parent_idx = (index - 1) // 2

    # If we are already at the root of the heap, return nothing
    if parent_idx < 0:
        return

    # If the current node is greater than its root, swap them
    if heap[index] > heap[parent_idx]:
        _swap(heap, index, parent_idx)
        _bottom_up(heap, parent_idx)  # iteratively call bottom_up


    # private: heapify, for ensuring heap is in order after root is popped
def _top_down(heap, index):
    # Finding parent of the element
    child_idx = 2 * index + 1

    # If we are at the end of the heap, return nothing
    if child_idx >= len(heap):
        return

    # swap with the larger one of two children
    if child_idx + 1 < len(heap) and heap[child_idx] < heap[child_idx + 1]:
        child_idx += 1

    # If the child node is smaller than the current node, swap them
    if heap[child_idx] > heap[index]:
        _swap(heap, child_idx, index)
        _top_down(heap, child_idx)
```

5. 初始化 I, P 以及 A：
   - I: 用來記錄文件是否被合併（還活著），0 代表已被合併的文件，1 為還沒被合併過，或者是合併別人的文件
   - P: 初始化陣列以類別化成 Max Heap
   - A: 紀錄每次 iteration 合併的是哪兩個文件

```
'''Initialize I[] for checking alive, default is set to 1'''
I = [1] * 1096

'''Initilaize P[]'''
P = [0] * 1096

for i in range(1, 1096):
    P[i] = Max_Heap(C[i])
    P[i].pop()  # remove itself which cosine similarity will definitely be 1.0

'''Initialize A[] for recording set of merged documents'''
A = []
```

6. 找出一組組一樣的文件

在做作業的過程中，發現有些文件除了跟自己的相似度是 1 以外，也有跟其他文件出現相似度為 1 的情況，我認為在做全部文件分群之前應該先手動將一樣的文件分到同一群，再繼續往下做。

```
'''Find sets of news data that have same content^_^;'''
twins_temp = []

for i in range(1, 1096):
    if(P[i].peek() == 1):  # find whose similarity value is 1 in P[i]
        twins_temp.append(i)  # store into twins_temp

twins = []
for i in twins_temp:
    temp = []
    for j in twins_temp:  # find out group of data that have same content
        if(cosine(i,j) == 1):  # (承上註解) through cosine similarity equals to 1
            temp.append(j)
    twins.append(sorted(temp))

twins_set = set(tuple(l) for l in twins)
# twins_set
```

於是剛開始先合併這些 cosine similarity 彼此為 1 的文件們，下圖範例為兩兩一組相同的文件，之後還有處理三個一組、四個一組相同的文件。

```
'''Merge 17 groups of same data^_^;'''

count_twins = 0

for twin in twins_set:
    k1, k2 = twin[0], twin[1]
    A.append([k1, k2])
    count_twins += 1
    I[k2] = 0
    P[k1] = Max_Heap([])

    for i in range(1, 1096):
        if(I[i]==1 and i!=k1 ):
            P[i].delete(C[i][k1])  # 去 i 的 Max Heap 刪掉跟 k1 有關的 node
            P[i].delete(C[i][k2])  # 去 i 的 Max Heap 刪掉跟 k2 有關的 node

            C[i][k1] = max(C[i][k1], C[i][k2])  # 挑大的
            P[i].push(C[i][k1])  # 重新計算每棵 k1 以外的 Max Heap

            C[k1][i] = C[i][k1]
            P[k1].push(C[k1][i])  # 重新長出 k1 的 Max Heap

    if(len(twin) >= 3): (文件略)
```

7. 處理剩餘的一千多篇文件

```
'''HAC: merge the rest of documents'''
for _iter in range(1095-count_twins-1):
    # print(_iter)

    max_sim = 0
    max_id = [0, 0]

    # pick the current set of doc. that have maximal similarities through all heaps
    for i in range(1, 1096):
        if(I[i]):
            if(P[i].peek() > max_sim):  # find the maximal similarity
                max_sim = P[i].peek()
                max_id[0] = i
            if(P[i].peek() == max_sim):  # find its partner
                max_id[1] = i

    # merge the current set of doc. that have largest similarity
    k1, k2 = max_id[0], max_id[1]
    A.append([k1, k2])
    I[k2] = 0
    P[k1] = Max_Heap([])

    for i in range(1, 1096):
        if(I[i]==1 and i!=k1 ):
            P[i].delete(C[i][k1])  # 去 i 的 Max Heap 刪掉跟 k1 有關的 node
            P[i].delete(C[i][k2])  # 去 i 的 Max Heap 刪掉跟 k2 有關的 node

            C[i][k1] = max(C[i][k1], C[i][k2])  # 挑大的
            P[i].push(C[i][k1])  # 重新計算每棵 k1 以外的 Max Heap

            C[k1][i] = C[i][k1]
            P[k1].push(C[k1][i])  # 重新長出 k1 的 Max Heap
```

由於老師在課堂提到新聞分群通常會使用 single-link，本次作業選擇用 single-link 實作。

8. 最後要輸出結果時，用以下兩個 function 來處理：

8.1. merge_list: 用聯集的概念整理 list (A)，確保不再出現重複的 item

```
def merge_list(L):  # 用聯集的概念整理 A
    '''
    example
    A=[[1,2], [3,4], [5,6], [1,6]]
    A_set = [set(i) for i in A]
    >> [{1, 2}, {3, 4}, {5, 6}, {1, 6}]
    merge_list(A_set)
    >> [{3, 4}, {1, 2, 5, 6}]
    '''
    length = len(L)
    for i in range(1, length):
        for j in range(i):
            if L[i] == {0} or L[j] == {0}:
                continue
            x = L[i].union(L[j])
            y = len(L[i]) + len(L[j])
            if len(x) < y:
                L[i] = x
                L[j] = {0}

    return [i for i in L if i != {0}]
```

8.2. output: 輸入 list 以及 k，例如要分成 1094 群就代表 k=1094，只要做 N-k = 1095-1094 = 1 次合併。最後整理成要輸出的樣子。

```
def output(A, k):
    A_set = [set(i) for i in A[:(1095-k)]]  # cut the A list at (1095-k) step for k clusters
    merged_list = merge_list(A_set)  # 把 A 整理成沒有重複 group 的 list
    result = merged_list  # store final result

    for i in range(1, 1096):
        tag = False  # default: item is not in the merged list
        for item_set in merged_list:
            if i in item_set:
                tag = True
        if tag == False:
            result.append(set([i]))  # 落單的人 append 在最後面
    return result
```

9. 分群結果：將結果照順序 output 成 txt file

```
final_20 = output(A, 20)
final_20_path = "./20.txt"
f = open(final_20_path, 'w')

for cluster in final_20:
    for member in sorted(cluster):
        f.write(str(member)+"\n")
    f.write("\n")
f.close()
```

10. Heap 實作時的參考資料：

```
'''
Reference
http://www.mathcs.emory.edu/~cheung/Courses/171/Syllabus/9-BinTree/heap-delete.html
https://medium.com/@Kadai/%E8%B3%87%E6%96%99%E7%B5%90%E6%A7%8B%E5%A4%A7%E4%BE%BF%E7%95%B6-binary-heap-ec47ca7aebac
'''
```