Information Retrival Programming Assignment#4
B06406009 資管三 陳姵如

A. 開發環境
Mac OS / Jupyter notebook

B. 實作 Heap

In [423]:
```
#Heap Algorithm
class Heap(object):
    def __init__(self):
        self.__array = []
        self.__last_index = -1

    def size(self):
        return len(self.__array)


    def push(self, value):
        self.__array.append(value)
        self.__last_index += 1
        self.__siftup(self.__last_index)

    def delete(self, index):
        if self.__last_index == -1:
            raise IndexError("Can't pop from empty heap")
        root_value = self.__array[index]
        if self.__last_index > 0:  # more than one element in the heap
            self.__array[index] = self.__array[self.__last_index]
            self.__siftdown(index)
        self.__last_index -= 1
        return root_value

    def pop(self):
        if self.__last_index == -1:
            raise IndexError("Can't pop from empty heap")
        root_value = self.__array[0]
        if self.__last_index > 0:  # more than one element in the heap
            self.__array[0] = self.__array[self.__last_index]
            self.__siftdown(0)
        self.__last_index -= 1
        return root_value

    def peek(self):
        if not self.__array:
            return None
        return self.__array[0]

    def replace(self, new_value):
        if self.__last_index == -1:
            raise IndexError("Can't pop from empty heap")
        root_value = self.__array[0]
        self.__array[0] = new_value
        self.__siftdown(0)
        return root_value

    def heapify(self, input_list):
        n = len(input_list)
        self.__array = input_list
        self.__last_index = n-1
        for index in reversed(range(n//2)):
            self.__siftdown(index)

    @classmethod
    def createHeap(cls, input_list):
        heap = cls()
        heap.heapify(input_list)
        return heap

    def __siftdown(self, index):
        current_value = self.__array[index]
        left_child_index, left_child_value = self.__get_left_child(index)
        right_child_index, right_child_value = self.__get_right_child(index)
        best_child_index, best_child_value = (right_child_index, right_child_value) if
        is not None and self.comparer(right_child_value, left_child_value) else (left_c
        if best_child_index is not None and self.comparer(best_child_value, current_val
            self.__array[index], self.__array[best_child_index] =\
                best_child_value, current_value
            self.__siftdown(best_child_index)
        return
```

加了一層 interface “MaxHeap” ，創建時使用 MaxHeap

```
class MaxHeap(Heap):
    def comparer(self, value1, value2):
        return value1 > value2
```

把每群 / 文件 互相的 cosine similarity 存成 MaxHeap，因為每次需求最大。
以下為主程式，解釋於註解部分。
使用 centroid clustering。

In [*]:
```
K20 = 20
docID = -1
peer = -1
clustering_count = 0
merged = []
merged_recorad = []
tmp = []

while clustering_count < K20: #收斂到剩下20

    maxcos = -1

    for i in range(len(cosineTable2)): #len = 1095
        try: #為了解決none
            current = cosineTable2[i].peek()
            if current[1] in merged:
                cosineTable2[i].pop()
#                 print("current_pop",i , current)
            else:
                for i in range(len(cosineTable2)):
                    if current[0] > maxcos: #選出1095個中 cosine similarity 最大
                        maxcos = current[0]
                        docID = i
                        peer = current[1]
        except:
            continue

    if docID not in merged:
        merged.append(docID)
    if peer not in merged:
        merged.append(peer)
    clustering[docID].append(peer)

    #更新vector[docID]
    for i in range(len(vector2[docID])):
        try:
            vector2[docID][i] = (vector2[docID][i]*(len(clustering[docID])) + vector2[p
        except:
            pass

    #更新cosineTable[docID]
    for i in range(cosineTable2[docID].size()):
        k = docID + i + 1
        try: #有些vector被刪掉
            cosineTable2[docID][i].delete(i)
            cosineTable2[docID][i].push((cosine(vector[docID],vector[k],k)))
        except:
            cosineTable2[docID].delete(i)
            cosineTable2[docID][i].push((cosine(-1,k)))


    #刪除peer vector/clustring
    del cosineTable2[peer] #刪除peer cosinTable
    cosineTable2.insert(peer,tmp)
    del vector2[peer]
    vector2.insert(peer,tmp)

    for i in range(len(clustering[peer])-1):
        if clustering[peer][i] not in merged:
            merged.append(clustering[peer][i])

    del clustering[peer]
    clustering.insert(peer,tmp)

    clustering_count = 0
    for i in range(len(clustering)):
        if len(clustering[i]) != 0:
            print(len(clustering[i]))
            clustering_count = clustering_count + 1
```