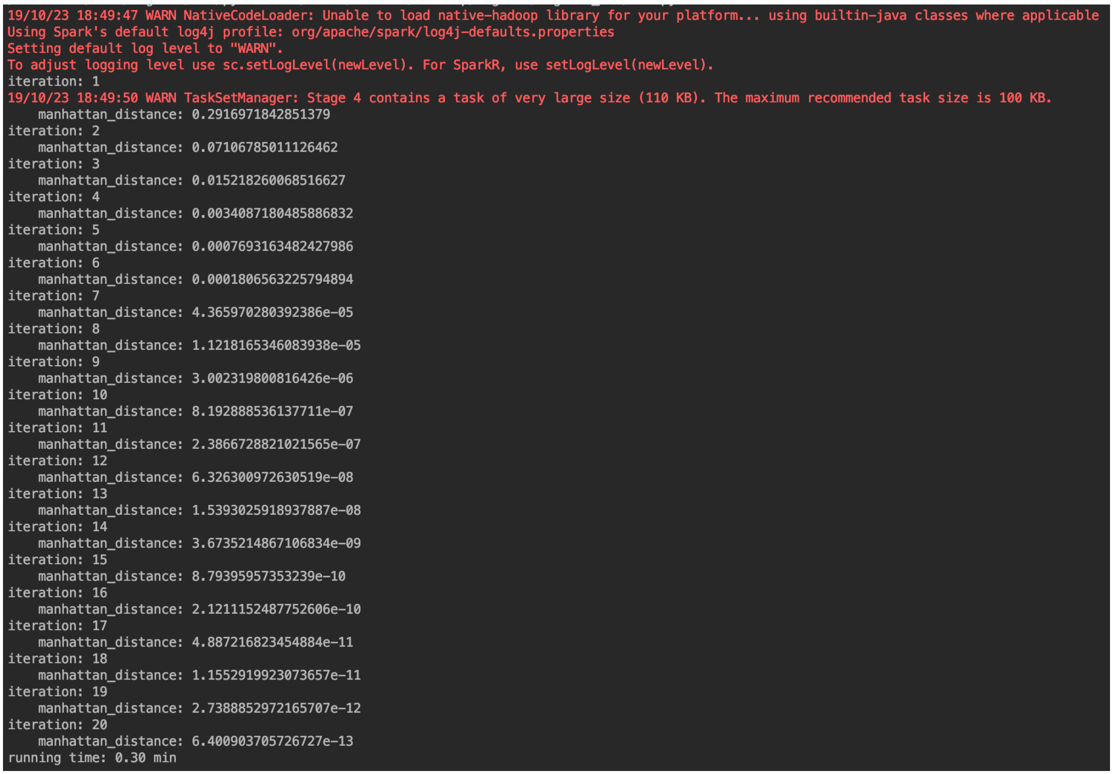
巨量資料 HW2

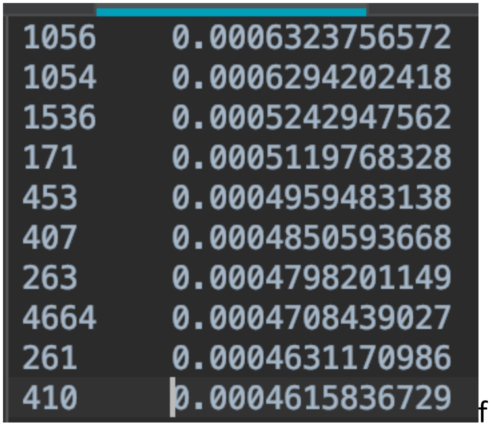
108062566

資工碩一 陳法佑

a-1. iterate 20 次，運行時間 0.3 min



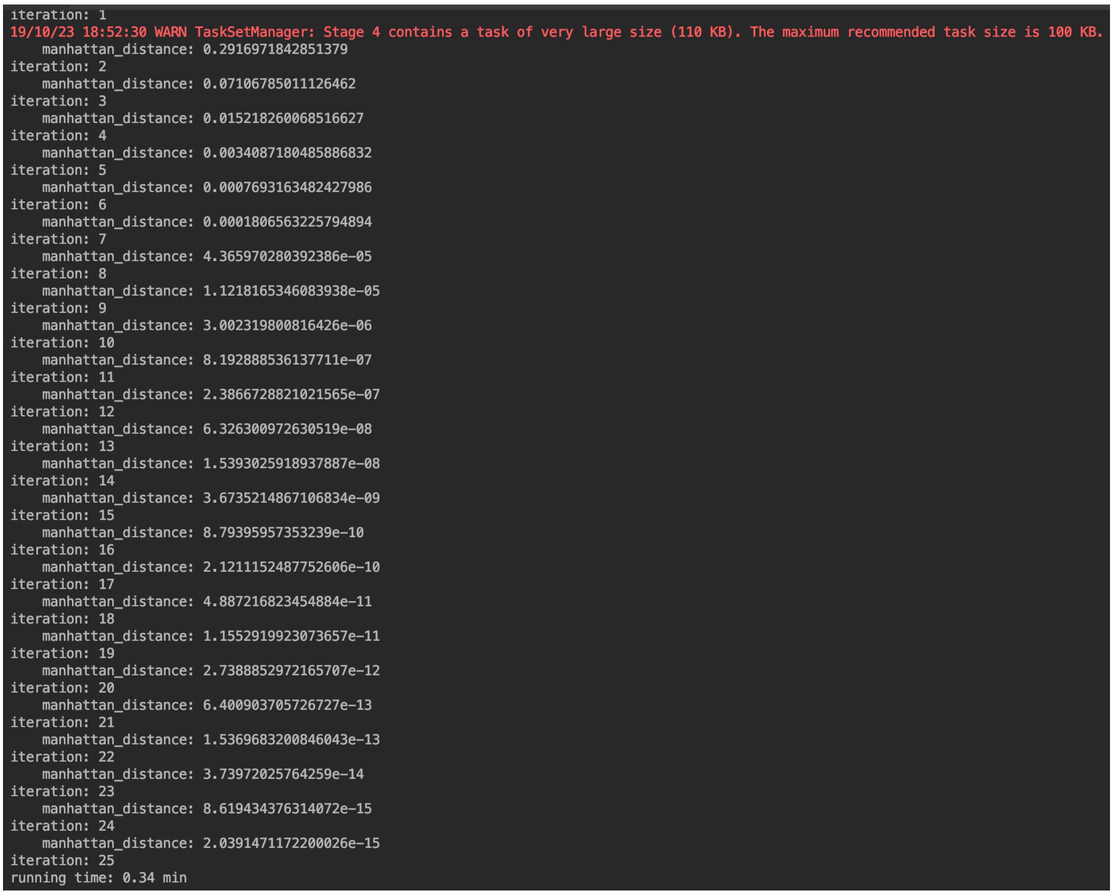
```
19/10/23 18:49:47 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
iteration: 1
19/10/23 18:49:50 WARN TaskSetManager: Stage 4 contains a task of very large size (110 KB). The maximum recommended task size is 100 KB.
    manhattan_distance: 0.2916971842851379
iteration: 2
    manhattan_distance: 0.07106785011126462
iteration: 3
    manhattan_distance: 0.015218260068516627
iteration: 4
    manhattan_distance: 0.0034087180485886832
iteration: 5
    manhattan_distance: 0.0007693163482427986
iteration: 6
    manhattan_distance: 0.0001806563225794894
iteration: 7
    manhattan_distance: 4.365970280392386e-05
iteration: 8
    manhattan_distance: 1.1218165346083938e-05
iteration: 9
    manhattan_distance: 3.002319800816426e-06
iteration: 10
    manhattan_distance: 8.192888536137711e-07
iteration: 11
    manhattan_distance: 2.3866728821021565e-07
iteration: 12
    manhattan_distance: 6.326300972630519e-08
iteration: 13
    manhattan_distance: 1.5393025918937887e-08
iteration: 14
    manhattan_distance: 3.6735214867106834e-09
iteration: 15
    manhattan_distance: 8.79395957353239e-10
iteration: 16
    manhattan_distance: 2.1211152487752606e-10
iteration: 17
    manhattan_distance: 4.887216823454884e-11
iteration: 18
    manhattan_distance: 1.1552919923073657e-11
iteration: 19
    manhattan_distance: 2.7388852972165707e-12
iteration: 20
    manhattan_distance: 6.400903705726727e-13
running time: 0.30 min
```



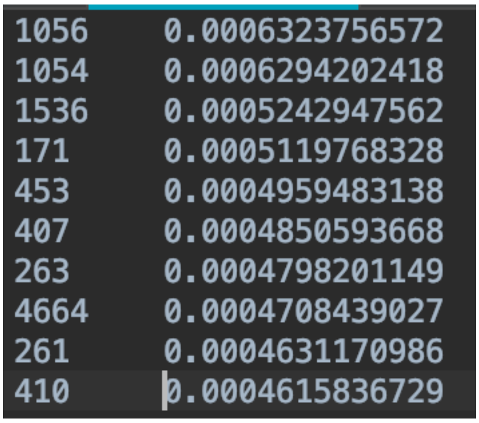
```
1056    0.0006323756572
1054    0.0006294202418
1536    0.0005242947562
171     0.0005119768328
453     0.0004959483138
407     0.0004850593668
263     0.0004798201149
4664    0.0004708439027
261     0.0004631170986
410     0.0004615836729
```

f

a-2. iterate 至收斂 (使用曼哈頓距離，epilson=1e-15)，iterate 24 次，運行時間 0.34 min

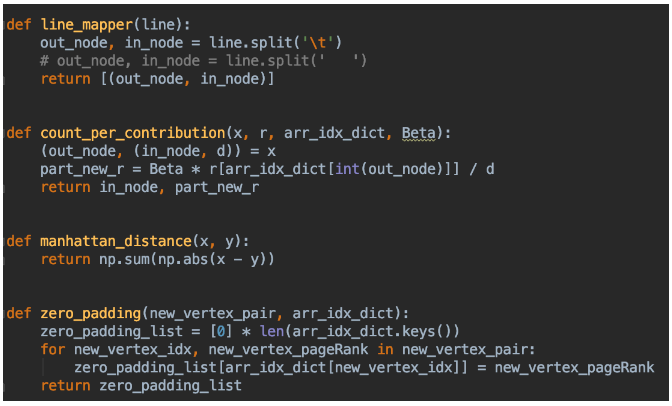


```
iteration: 1
19/10/23 18:52:30 WARN TaskSetManager: Stage 4 contains a task of very large size (110 KB). The maximum recommended task size is 100 KB.
    manhattan_distance: 0.2916971842851379
iteration: 2
    manhattan_distance: 0.07106785011126462
iteration: 3
    manhattan_distance: 0.015218260068516627
iteration: 4
    manhattan_distance: 0.0034087180485886832
iteration: 5
    manhattan_distance: 0.0007693163482427986
iteration: 6
    manhattan_distance: 0.0001806563225794894
iteration: 7
    manhattan_distance: 4.365970280392386e-05
iteration: 8
    manhattan_distance: 1.1218165346083938e-05
iteration: 9
    manhattan_distance: 3.002319800816426e-06
iteration: 10
    manhattan_distance: 8.192888536137711e-07
iteration: 11
    manhattan_distance: 2.3866728821021565e-07
iteration: 12
    manhattan_distance: 6.326300972630519e-08
iteration: 13
    manhattan_distance: 1.5393025918937887e-08
iteration: 14
    manhattan_distance: 3.6735214867106834e-09
iteration: 15
    manhattan_distance: 8.79395957353239e-10
iteration: 16
    manhattan_distance: 2.1211152487752606e-10
iteration: 17
    manhattan_distance: 4.887216823454884e-11
iteration: 18
    manhattan_distance: 1.1552919923073657e-11
iteration: 19
    manhattan_distance: 2.7388852972165707e-12
iteration: 20
    manhattan_distance: 6.400903705726727e-13
iteration: 21
    manhattan_distance: 1.5369683200846043e-13
iteration: 22
    manhattan_distance: 3.73972025764259e-14
iteration: 23
    manhattan_distance: 8.619434376314072e-15
iteration: 24
    manhattan_distance: 2.0391471172200026e-15
iteration: 25
running time: 0.34 min
```



```
1056    0.0006323756572
1054    0.0006294202418
1536    0.0005242947562
171     0.0005119768328
453     0.0004959483138
407     0.0004850593668
263     0.0004798201149
4664    0.0004708439027
261     0.0004631170986
410     0.0004615836729
```

b. Explain how you design your mapper and reducer.

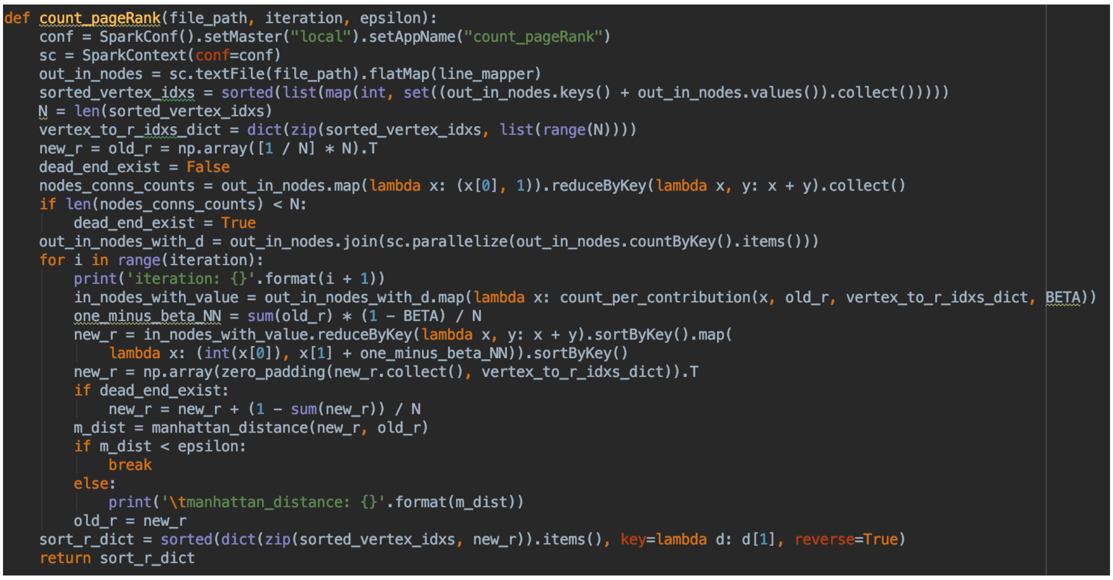


```python
def line_mapper(line):
    out_node, in_node = line.split('\t')
    # out_node, in_node = line.split('    ')
    return [(out_node, in_node)]


def count_per_contribution(x, r, arr_idx_dict, Beta):
    (out_node, (in_node, d)) = x
    part_new_r = Beta * r[arr_idx_dict[int(out_node)]] / d
    return in_node, part_new_r


def manhattan_distance(x, y):
    return np.sum(np.abs(x - y))


def zero_padding(new_vertex_pair, arr_idx_dict):
    zero_padding_list = [0] * len(arr_idx_dict.keys())
    for new_vertex_idx, new_vertex_pageRank in new_vertex_pair:
        zero_padding_list[arr_idx_dict[new_vertex_idx]] = new_vertex_pageRank
    return zero_padding_list
```



```python
def count_pageRank(file_path, iteration, epsilon):
    conf = SparkConf().setMaster("local").setAppName("count_pageRank")
    sc = SparkContext(conf=conf)
    out_in_nodes = sc.textFile(file_path).flatMap(line_mapper)
    sorted_vertex_idxs = sorted(list(map(int, set((out_in_nodes.keys() + out_in_nodes.values()).collect()))))
    N = len(sorted_vertex_idxs)
    vertex_to_r_idxs_dict = dict(zip(sorted_vertex_idxs, list(range(N))))
    new_r = old_r = np.array([1 / N] * N).T
    dead_end_exist = False
    nodes_conns_counts = out_in_nodes.map(lambda x: (x[0], 1)).reduceByKey(lambda x, y: x + y).collect()
    if len(nodes_conns_counts) < N:
        dead_end_exist = True
    out_in_nodes_with_d = out_in_nodes.join(sc.parallelize(out_in_nodes.countByKey().items()))
    for i in range(iteration):
        print('iteration: {}'.format(i + 1))
        in_nodes_with_value = out_in_nodes_with_d.map(lambda x: count_per_contribution(x, old_r, vertex_to_r_idxs_dict, BETA))
        one_minus_beta_NN = sum(old_r) * (1 - BETA) / N
        new_r = in_nodes_with_value.reduceByKey(lambda x, y: x + y).sortByKey().map(
            lambda x: (int(x[0]), x[1] + one_minus_beta_NN)).sortByKey()
        new_r = np.array(zero_padding(new_r.collect(), vertex_to_r_idxs_dict)).T
        if dead_end_exist:
            new_r = new_r + (1 - sum(new_r)) / N
        m_dist = manhattan_distance(new_r, old_r)
        if m_dist < epsilon:
            break
        else:
            print('\tmanhattan_distance: {}'.format(m_dist))
        old_r = new_r
    sort_r_dict = sorted(dict(zip(sorted_vertex_idxs, new_r)).items(), key=lambda d: d[1], reverse=True)
    return sort_r_dict
```

1. 先讀取 file，並切割成 out_nodes、in_nodes。

2. 將 out_nodes、in_nodes 建成一個從小到大排序的 set，sorted_vertex_idxs，它的長度便是 N 的值。

3. 因為 nodes 的號碼並不是完全按照順序，有時候會跳號，所以建一個 dict，vertex_to_r_idxs_dict，紀錄每個 vertex 真正對應到 r 陣列的哪個位置。

4. 初始化 r 陣列。

5. 計算每個 nodes 的 out_connections，如果這些 nodes 的數量小於 N，表示有 node 的 out_connection 為 0，表示存在 dead_end。

6. out_in_nodes_with_d: 其格式為 (out_node, (in_node, d))，key 為 out_node，因為我們之後在計算 contribution 的時候，同一個 out_node 向外

連結的 M 矩陣值， 1 / d， 要乘以同一個 r，其中 d 為這個 out_node 向外連結的數量。而 d 的取法是利用 pyspark 提供的 countByKey()，計算每個 key 的數量。

7. 接下來要進入 iterate 迴圈，要計算 new_r 的值，計算方法為將 out_in_nodes_with_d 中的每個元素，個別計算 Beta * r / d 的值，值得注意的是 r 的值要根據 r[arr_idx_dict[int(out_nodes)]] 獲得， arr_idx_dict 為第三點提的 vertex_to_r_idxs_dict。最後此 map 回傳格式 (in_node, part_new_r)。

8. 計算 (1 – Beta) / N * (all r)，因為 (1 - Beta) / N 為定值，所以可以把公式改寫(1 - Beta) / N * sum (r)，之後再把此值加回 all r。

9. 根據第 (7) 點的輸出，把同一個 in_node，也就是同一個 key 對應到的 part_new_r 相加，就是我們要的 new_r。接著再利用 pyspark 的 sortByKey() 讓 key 從小到大排序，接著再利用 map 將對所有 new_r 元素加上第 (8) 點的輸出。值得一提的是，此時 new_r 的元素不一定與 N 相同，原因是有些 nodes 只有 out connections 而沒有 in connections，這種情況下這些 nodes 並不會有 pageRank，因此不會出現在 new_r 的元素裡。此時實作 zero_padding 的 function，將這些沒有 pageRank 的 nodes 對應到 arr_idx_dict 的位置補 0，讓維度維持在(N, 1)。接下來就只剩下處理 dead_end 發生的情況。

10. 根據第 (5) 點輸出，如果存在 dead_end，則對所有 new_r 加上 (1 - s) / N。

11. 計算曼哈頓距離，如果此距離小於 epilson (1e-15)，則表示收斂。

12. 收斂時將 new_r 根據 value 排序並存成 dict，最後再寫入 .txt 檔。