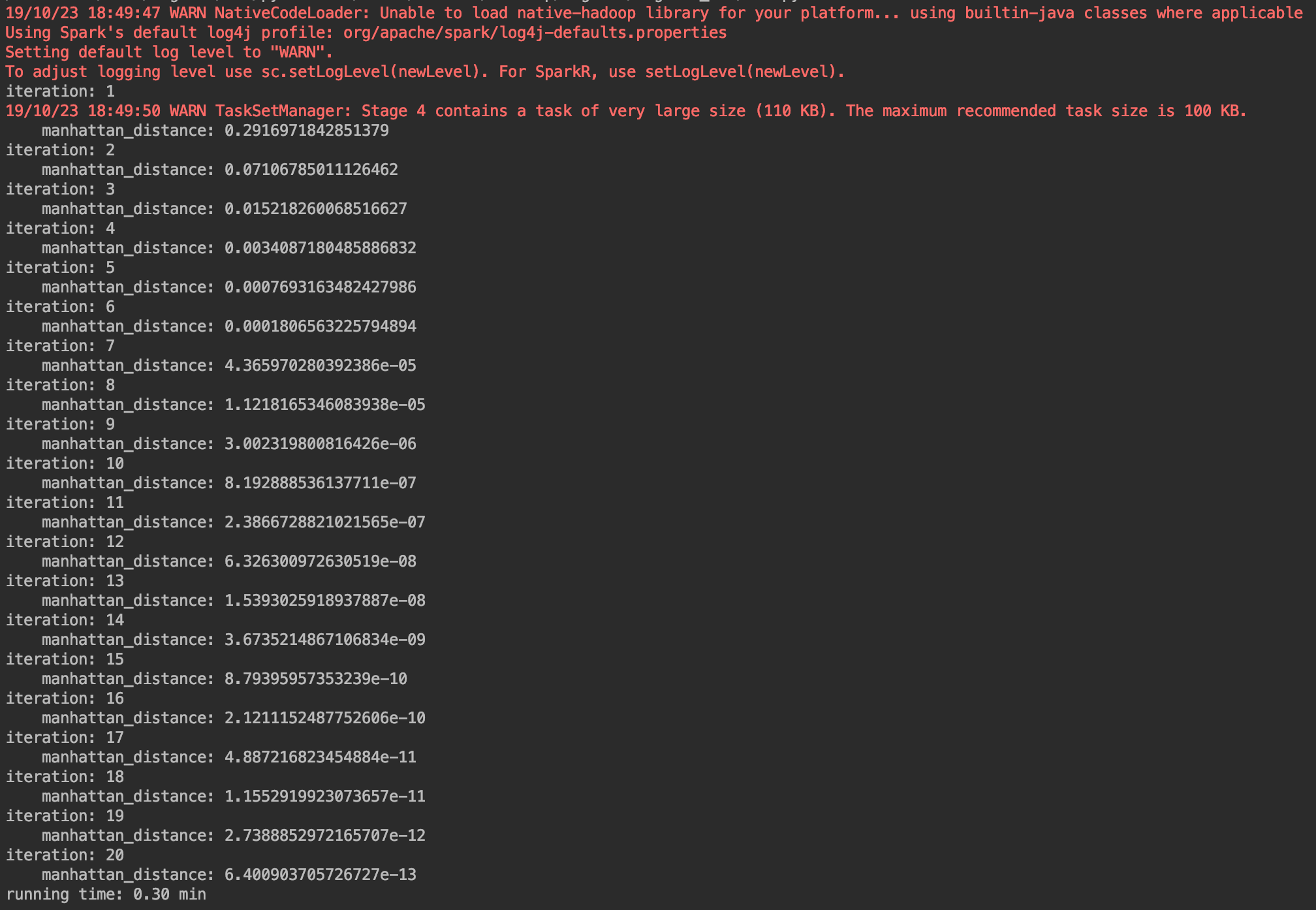
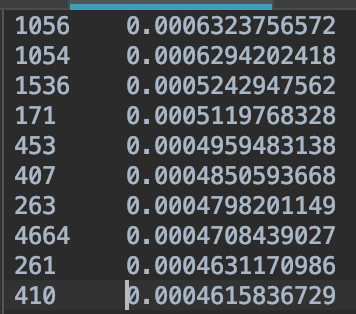
巨量資料 HW2

108062566

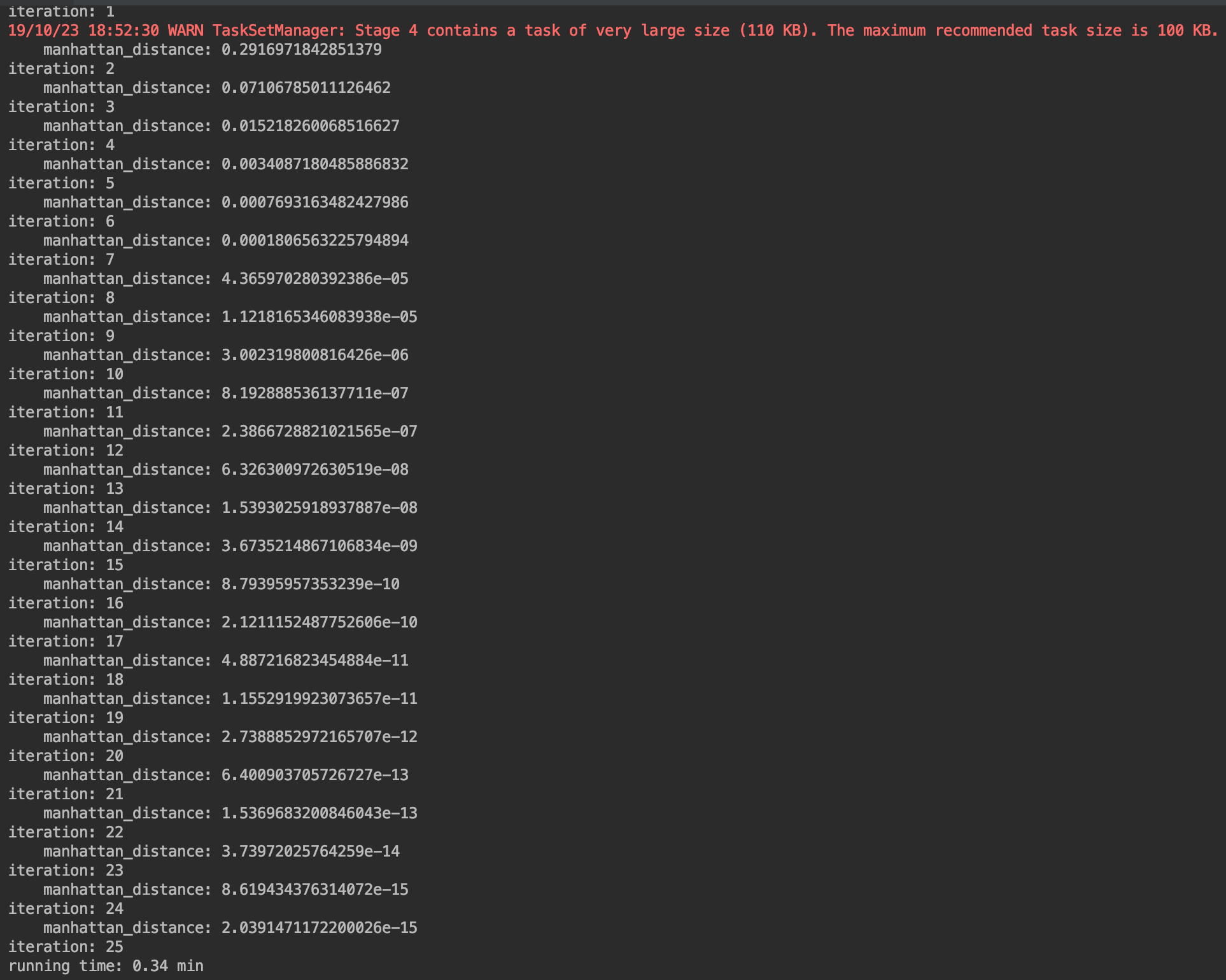
資工碩一 陳法佑

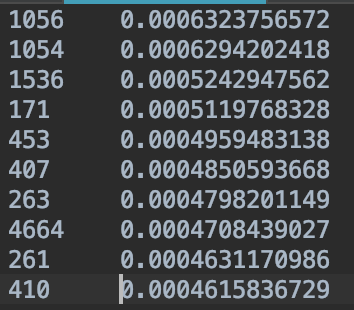
a-1. iterate 20次，運行時間 0.3 min

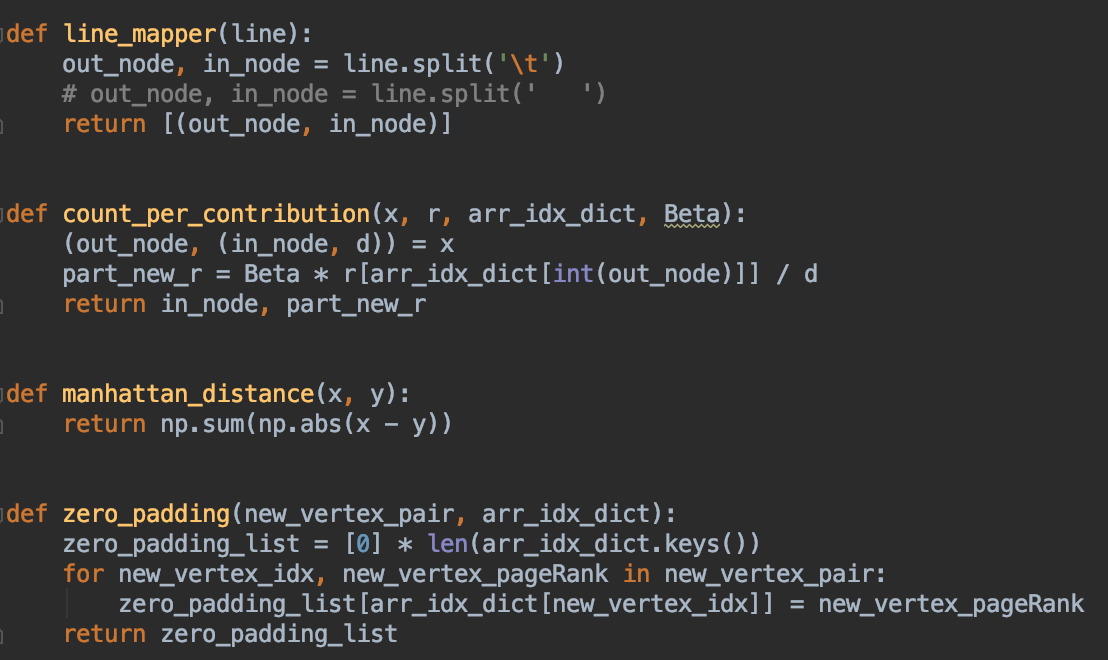


f

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





b. Explain how you design your mapper and reducer.  




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。