# PageRank

103062318 蔡尚倫

## Final output. (We require 20 iterations result)：

1056 0.000632138

1054 0.000629095

1536 0.000523847

171 0.000511561

453 0.000495598

407 0.000484785

263 0.000479561

4664 0.000470442

261 0.000462835

410 0.000461454

因為在考慮計算方便，使用四捨五入，因此可能會丟棄掉一些資訊，所以後面的數值也相對捨去。在PDF給的範例中，確實能達到範例的需求。(PS：範例中是從NODE 1開始，與此次input不同)

## Code Structure：

MyEntry.java -> 仿照 HashMap 中的 Entry 所創的資料結構，用於最後排序動作

MDA_HW2.java -> main function(控制 MapReduce 的順序)以及相關資料存在 path，初始化 PageRank 初始值(使用 HashSet 先過濾一次 Input，得到真正存在的點後才做初始)，創建 List<MyEntry>，實作 Collections.sort，制定排序的規則， 讓 List 以 pagerank value 作為排序基礎，寫回 hadoop fs

MDA_HW2_Adj.java -> 將 input 點對點資料轉成矩陣形式，就是 Big Matrix M

MDA_HW2_PageMul.java -> 將 Big Matrix M 讀取與 PageRank 讀取，利用上次 MDA_HW1 矩陣相乘的架構修改

MDA_HW2_compen.java -> 由於有 Deadend node，所以需要有補償機制，方式就是 (1-sum of pagerank) / page number

clean.sh -> 因為 hadoop 無法覆蓋原本相同名稱的資料，所以每 run 完一次所有的 code，下次如果想要修改參數再 run 時，需要把部分資料夾內容刪除，由於重複性很高，就寫了個 sh 檔來做指令整合

Note:

1. 由於我在測試時是分開的，所以 page number 與 Beta 需要分開來設定，要確認所有 code 裡面所輸入的參數相同，而且我 page 所設定的參數為 MAX(出現過的 NODE)+1，在這次的作業裡 page 為 10879

2. 每次計算完各個 pagerank value 後，四捨五入至小數點下第九位，意即在小數點下第十位做四捨五入的動作，使用 DecimalFormat 來完成
3. 由於 input 會有不連續的點，我的處理方式是將不存在的點也放入我的 pagerank 的矩陣裡，可是每次做完後記得歸 0 就可以完成了，不需要再多一次 MapReduce 做過濾

## MapReduce_Implement：

### MDA_HW2_Adj.java：

Mapper：

String [] matrix = value.toString().split("\t")　//原始資料是用 tab 鍵分開

Text Map_key = new Text(matrix[0])　//使指向他人的同一個點，擁有相同的 key

Text Map_value = new Text(matrix[1]) //value 為時指向的該點

EX：　(1　　3) (1　　5)　就會被送到同一個 Reducer

Reducer：

Sum =　計算 key 該點總共指向多少人

StringBuilder sb = new StringBuilder();

sb.append("M,"+i+","+key.toString()+","+(float)(flee_v+B*tmp[i]/sum));　　//利用 buffer 準備好相關參數寫入，利用公式算出矩陣上該點的數值

context.write(null,new Text(sb.toString()));　//將其包裝成 MDA_HW1 的矩陣形式，所以寫出的 key 是 null，才不會產生 key 與 value 間的 tab 空白

MDA_HW2_PageMul.java：(與 MDA_HW1 相同架構)

Mapper：

由於有兩個 input 所以需要 override Mapper 的 setup function 來判定是哪一個 input。

假如是剛產生 Adj Matrix

Map_key.set(mapAndreduce[1]+","+Integer.toString(i));

Map_value.set(mapAndreduce[0]+","+mapAndreduce[2]+","+mapAndreduce[3]);

context.write(Map_key,Map_value);

假如是 pagerank 輸入(需要給他 N 這個字串來連接 HW1 的格式)

Map_key.set(Integer.toString(i)+",1");

Map_value.set("N,"+mapAndreduce[0]+","+mapAndreduce[1]);

context.write(Map_key,Map_value);

Reducer：

與 HW1 大致相同(將 M、N 放進不同的 HashMap 中，在各自取出來做矩陣乘法，注意 Deadend 的處理在下方)，不一樣的地方在於因為的 Deadend node 的出現，所以必須補齊剛剛產生 Adj Matrix 中，Deadend node 的空缺 m_ij = m.containsKey(i) ? m.get(i):flee_v; 其中 flee_v 是(1-beta)/page num，因為 input 數據中沒有 Deadend，所以 containsKey 一定是空的，就必須取後面的 flee_v，最後再用 DecimalFormat 轉換 result，再 context.write 回去

MDA_HW2_compen.java：

Mapper：

由於要計算損失的量，需要把所有資料送到同一個 Reducer，才能在下一次的 Reducer 計算

String [] matrix = value.toString().split("\t");

Text Map_value = new Text(matrix[0]+","+matrix[1]);

context.write(new Text("a"),Map_value);

Reducer：(這裡注意 Iterable<Text>只能 Iterate 整個一次，理由網路上有)

兩個資料結構 List<String> K = new ArrayList<>();

List<Float> V = new ArrayList<>();在第一次 Iterate 中紀錄，之後計算完損失的總量後，平均加回每筆 page，用 DecimalFormat 轉換 result，

context.write(newText(K.get(i)),newText(df.format(result)));