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)));