**Cloud Programming Homework 3**

103062528 林玉山

**一、實作方式**

Step 1. 建製Graph

我將這個步驟拆成兩個phase，主要是為了找出不存在的node。第一個phase會分析輸入的file，找出裡面的title跟link，然後建立「反向的graph」。反向graph就是以link到的node當作source，然後把原本的source當作link，從圖像上來說就是把link的方向反過來。第二個phase再將前一步建出來的graph反建回去。

那我是如何找出不存在的node呢？在第一步建反向graph的時候，會特別加一條self-link，也就是link回自己。對於不存在的node來說，第一個phase就不會為那些node建出self-link。因此第二個phase只要找出沒有self-link的node，那就是不存在的點。

以下舉一個例子解釋：

假設現在有一筆這樣的資料



然後經過第一個phase會得到這樣的結果 (箭頭代表link)



我們可以很明顯地看出，D這個node並沒有在原本的資料中，但是建反向graph後就會出現在清單內。另外，我們有建self-link，但是D卻沒有出現self-link。因此只要忽略沒有self-link的node，就可以過濾掉不存在的node。

最後會產生出像是這樣的output file，page rank的初始值皆設為1.0，每個詞以tab字元隔開。



Step 2. 計算Page Rank

這步會開始計算page rank。在Mapper之中每個node會先將自己的page rank除以link數，然後將數值送給那些link到的node。同時也會將這個node的現在的page rank與link到的node list以自己為key送出。

Reducer則會依照initial value為1的page rank公式計算新的page rank，以及計算這次與上次的difference。每次計算後會得到這樣的output：



程式會依照給定的參數，執行本步驟數次。

Step 3. Sorting

這步就沒甚麼特別了。主要任務是依照page rank排序。Mapper會將title與page rank作為key，傳給reducer。Reducer則單純output而已。為了確保只輸出單一結果，所以只有設定一個reducer。因為MapReduce本來就會做排序的工作，所以只要將compare的方式寫正確就好。

Step 4, Step 5

這兩步因為私人事務過多的關係，就沒有繼續實作了。

**二、使用方式**

本程式碼需先使用compile.sh進行編譯，編譯完後會得到hw3.jar這個檔案。要執行這個程式，可以使用以下指令執行：

hadoop jar hw3.jar slmt.courses.cp.hw3.BuildPageRank [Input Path] [Output Path] [Iteration Times]

第一個參數為測試資料在HDFS上的位置

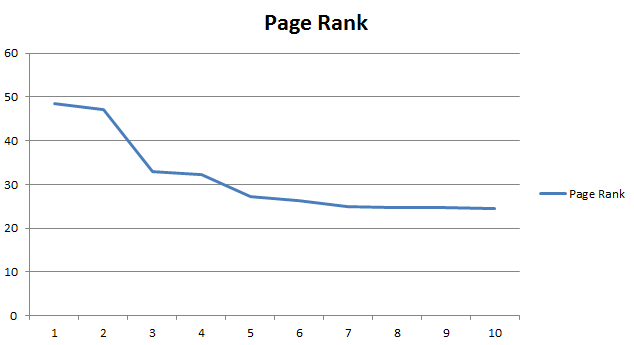
第二個參數為在HDFS上的輸出位置

第三個參數則用來設定第二步驟執行的次數

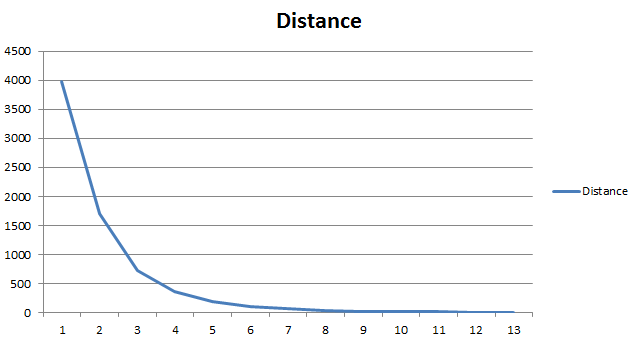
**三、實驗**

**1. 100M dataset**

Page Ranks

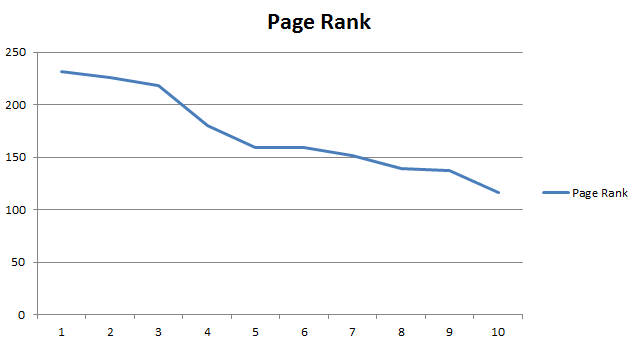


Iteration Distance

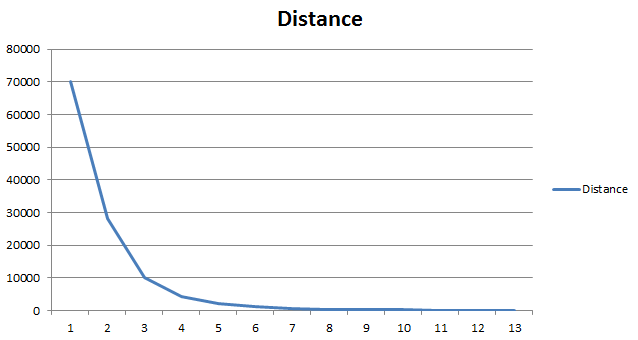


**2. 1G dataset**

Page Ranks

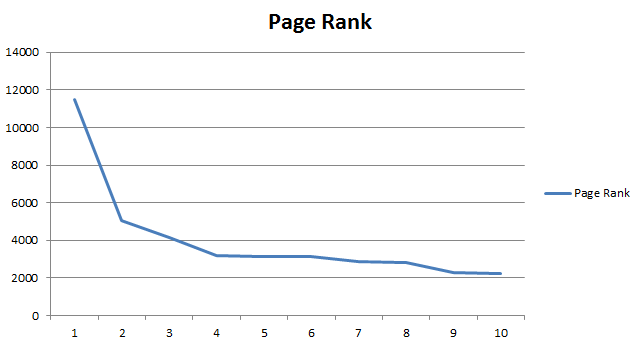


Iteration Distance



**3. 10G dataset**

Page Ranks



Iteration Distance



**四、實驗結果討論**

首先我們可以先看coverage rate。無論在100M, 1G或10G的資料中，可以發現差不多在第十次iteration時，page rank的變化就不大了。因此至少可以確定，在這些data set之中，iteration可能跑個10~20次左右就差不多了。

另外可以看到在100M, 1G的資料之中，前10名的Page Rank互相差距都差不多，不過在10G的資料裡，前10名的Page Rank差距就頗大。我自己猜測，少量的data set之中，不太容易出現很多page都link到的node。但是在大data set之中就容易出現。我覺得可能是當link一多，page rank就容易集中到少數幾個node上。因此才造成10G資料中，第一名特別高的狀況。