CSCI4180:Tutorial-7

Assignment 2 Review (Part 2)

Huancheng Puyang

Department of Computer Science & Engineering
The Chinese University of Hong Kong

2022.11.02

Outline

- Problem
  - Measure web page quality by PageRank algorithm
- Three main modules
  - PRPreProcess.java
  - PRNodeWritable.java
  - PageRank.java
- Implementation hints
- Submission

Outline

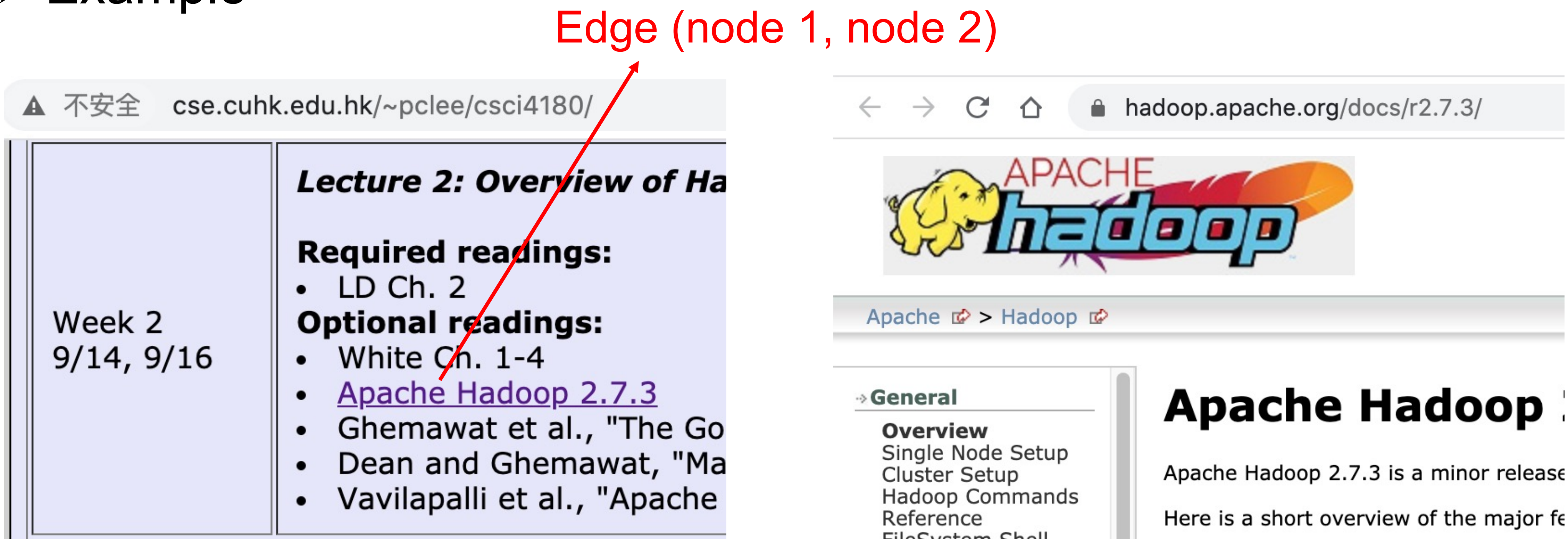
- Problem
  - Measure web page quality by PageRank algorithm
- Three main modules
  - PRPreProcess.java
  - PRNodeWritable.java
  - PageRank.java
- Implementation hints
- Submission

Problem

- Model of web pages
  - **Network of pages** are modeled as a **directed graph**
    - **Page** ➔ **node**
      - Each page is represented as a node
      - Each node takes a unique positive integer as the node ID
    - **Hyperlink** ➔ **edge**
      - When there is a hyperlink in page1 that leads user to page2, there is an edge from node1 to node2 in directed graph
- Dataset
  - A small one with a few lines and a large one sampled from Twitter dataset
  - Get it from Blackboard and use it for debug and test

Problem (Cont.)

- Example



Edge (node 1, node 2)

不安全 cse.cuhk.edu.hk/~pclee/csci4180/

Week 2 9/14, 9/16	***Lecture 2: Overview of Ha*** **Required readings:** • LD Ch. 2 **Optional readings:** • White Ch. 1-4 • Apache Hadoop 2.7.3 • Ghemawat et al., "The Go • Dean and Ghemawat, "Ma • Vavilapalli et al., "Apache

hadoop.apache.org/docs/r2.7.3/

Apache > Hadoop

General

Overview
Single Node Setup
Cluster Setup
Hadoop Commands Reference
FileSystem Shell

Apache Hadoop :

Apache Hadoop 2.7.3 is a minor release

Here is a short overview of the major fe

Node 1

Node 2

Problem (Cont.)

- PageRank algorithm
  - Measure the quality of web pages by **iterative** computation
- Simple sketch of algorithm (Slide 55 of lecture 5)
  - Caution: no dangling nodes and no random jump factor
  - $\boldsymbol{Page_i}$ starts with seed $\boldsymbol{PR_i}$ values (e.g., each page has equal PageRank, i.e., $\boldsymbol{PR_i = \frac{1}{|G|}}$)
  - $\boldsymbol{Page_i}$ distributes $\boldsymbol{PR_i}$ “credit” to all pages it links to
    - $\boldsymbol{Page_i}$ also receives credits from its predecessors by multiple in-bound links
  - $\boldsymbol{Page_i}$ adds up “credit” from multiple in-bound links to compute $\boldsymbol{PR_i{}'}$
  - Iterate until values converge

Problem (Cont.)

- ➢ Overview of part 3
  - Input format
    - Each line: <node ID 1> <node ID 2> <weight>
      - Similar as part 1, yet we **ignore weight** in part 2
      - Example: “5 2 14” means an edge from node 5 to node 2 **without weight**
    - Command line arguments
      - $iteration$: number of iterations for main loop (without any other stop condition)
      - $threshold$: minimal PageRank value for a node to output
      - $infile$: path of input file
      - $outdir$: path of output

Problem (Cont.)

- Overview of part 3 (Cont.)
  - Output format
    - Each line: <node ID> <PageRank value>
      - NOTE: you need to output all nodes whose PageRank value is above threshold

Problem (Cont.)

- Overview of part 3 (Cont.)
  - Pre-processing (PRPreProcess.java)
  - PageRank algorithm (PageRank.java)
    - Map: distribute the “credits” to all pages that the current page links to
    - Reduce: sum up the “credits” from all in-bound links
  - Output
    - Transform the results into required output format
  - Similar as workflow of part 2
    - Pre-processing (PDPreProcess.java)
    - ParallelDijkstra algorithm (ParallelDijkstra.java)
      - Map: emit the distances going through the current node
      - Reduce: update shortest distance according to received values
    - Output

Outline

- Problem
  - Measure web page quality by PageRank algorithm
- Three main modules
  - PRPreProcess.java
  - PRNodeWritable.java
  - PageRank.java
- Implementation hints
- Submission

Three Main Modules

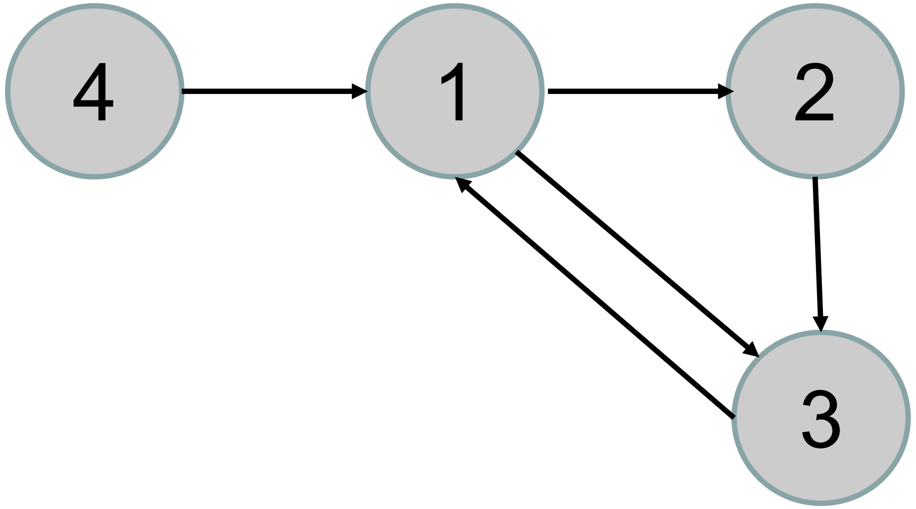
- PRPreProcess.java for parsing input
  - Parse input file into graph *G = (V, E)*
    - Represented by adjacency list
    - Similar as part 2 (see tutorial 6)
  - Simple example

4	1	1
1	2	1
1	3	1
2	3	1
3	1	1

Input

1	(2 3)
2	(3)
3	(1)
4	(1)

Adjacency list



Three Main Modules (Cont.)

- PRNodeWritable.java for node structure
  - For each node
    - Variable
      - Node ID
      - Adjacency list
      - PageRank value
      - …
    - Method
      - toString() and fromString()
      - readFields() and write()
  - Similar as PDNodeWritable.java in part 2 (see tutorial 6)
    - Both include node ID, adjacency list, and conversion methods
    - Difference: part 2 needs to store shortest distance and previous node

Three Main Modules (Cont.)

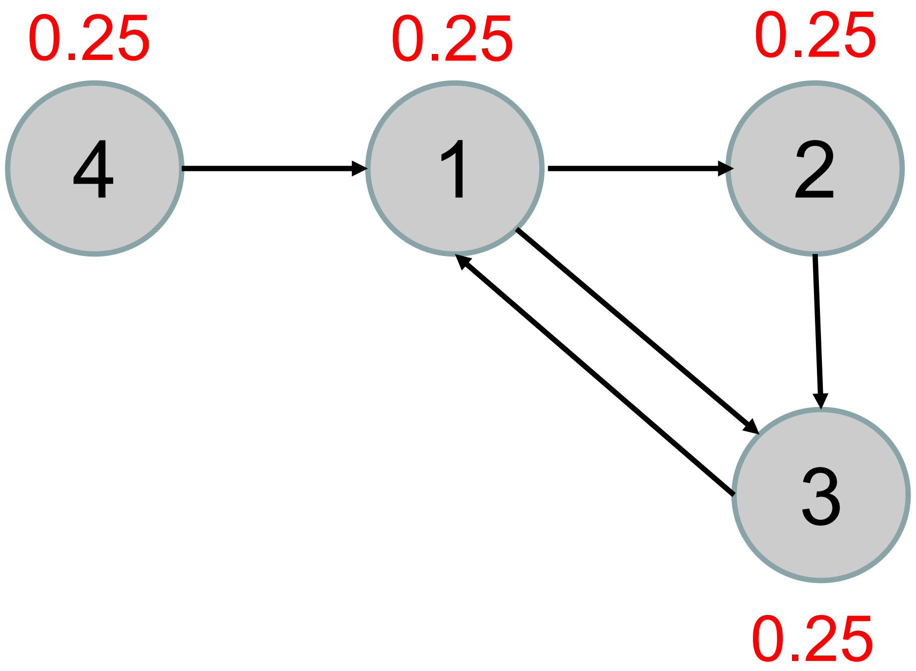
- PageRank.java for main module
  - Launch a MapReduce job for pre-processing module
  - **Main loop** to calculate per-node PageRank value iteratively
  - Cleanup for requirement output format
  - Similar as ParallelDijkstra.java in part 2
    - Launch a MapReduce job for pre-processing module
    - Main loop to update shortest distance by Dijkstra’s algorithm
    - Cleanup for requirement output format

Three Main Modules (Cont.)

- ➢ Main loop in PageRank.java (for each iteration)
  - MapReduce job to update PageRank value without random jump
    - Mapper
      - Get per-node PageRank value as input
        - For 1st iteration: initialize per-node PageRank value as 1/N, where N is # of nodes
        - Q: How to get the total number of nodes?
          - You can use Counters to count the number of nodes in pre-processing module

Three Main Modules (Cont.)

- Main loop in PageRank.java (for each iteration)
  - MapReduce job to update PageRank value without random jump
    - Mapper
      - Get per-node PageRank value as input
        - For 1st iteration: initialize per-node PageRank value as 1/N, where N is # of nodes

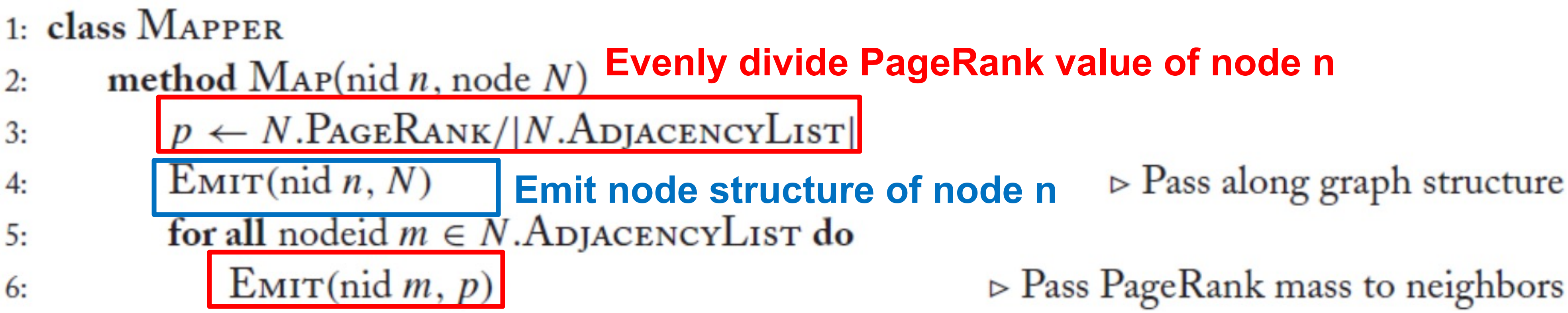


Three Main Modules (Cont.)

- ➢ Main loop in PageRank.java (for each iteration)
  - MapReduce job to update PageRank value without random jump
    - Mapper
      - Get per-node PageRank value as input
        - For 1st iteration: initialize per-node PageRank value as 1/N, where N is # of nodes
        - For other iterations: use results of previous iteration as the input of current iteration

Three Main Modules (Cont.)

- Main loop in PageRank.java (for each iteration)
  - MapReduce job to update PageRank value without random jump
    - Mapper
      - Get per-node PageRank value as input
      - Evenly distribute PageRank value of each node to its successors



```
class Mapper
    method Map(nid $n$, node $N$)
        $p \leftarrow N.\text{PageRank}/|N.\text{AdjacencyList}|$
        Emit(nid $n$, $N$)                                ▷ Pass along graph structure
        for all nodeid $m \in N$.AdjacencyList do
            Emit(nid $m$, $p$)                            ▷ Pass PageRank mass to neighbors
```

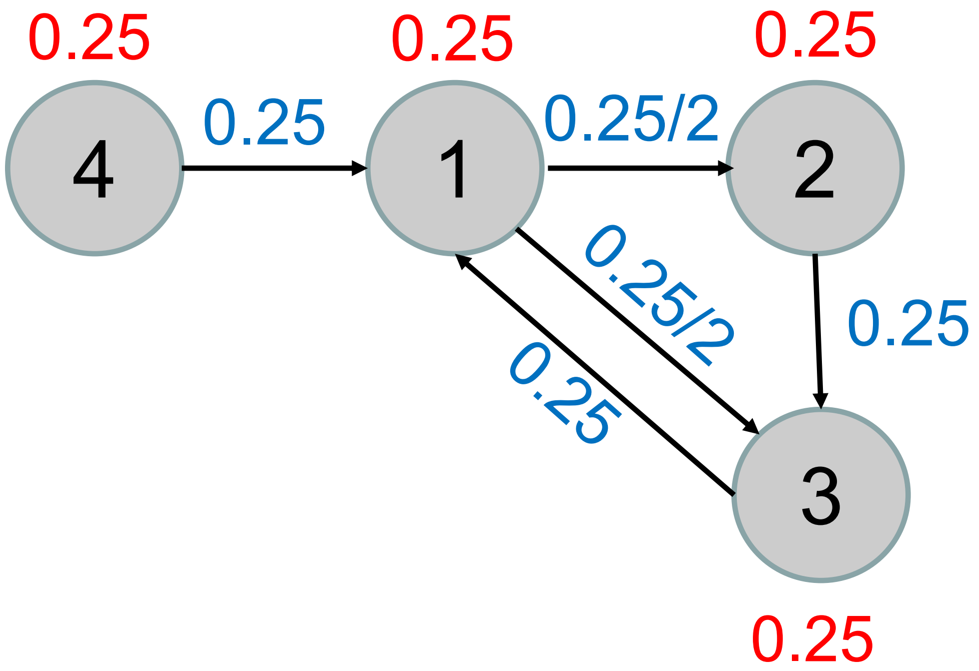
Evenly divide PageRank value of node n

Emit node structure of node n

Emit divided PageRank value to each successor

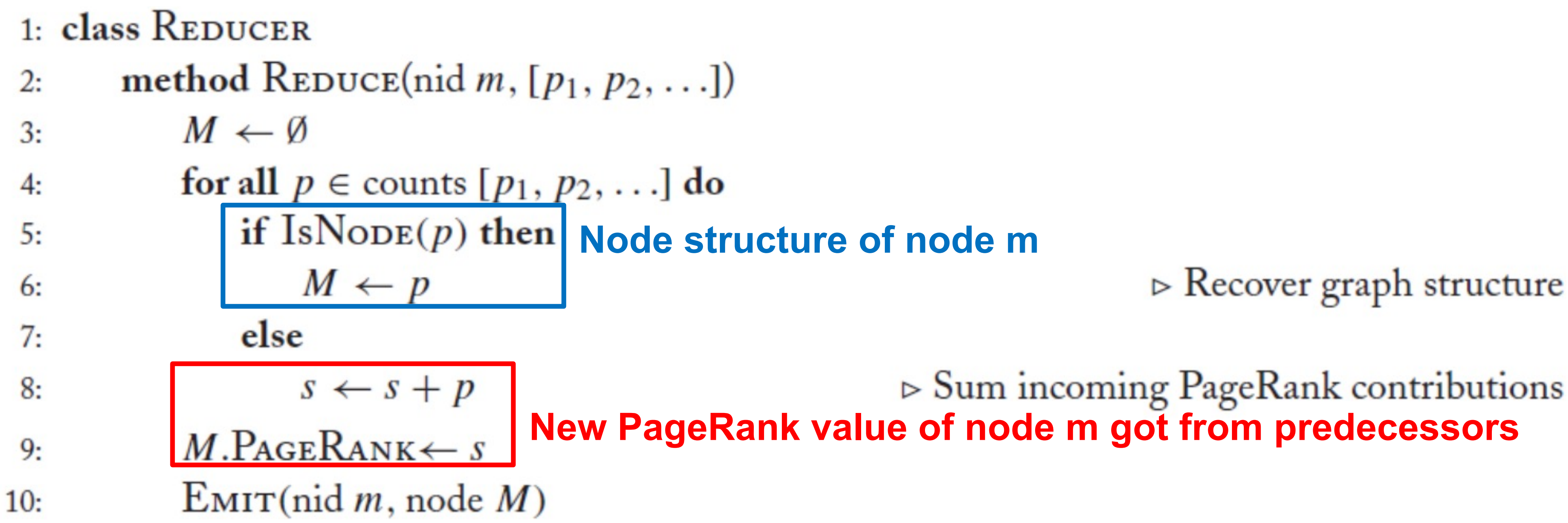
Three Main Modules (Cont.)

- Main loop in PageRank.java (for each iteration)
  - MapReduce job to update PageRank value without random jump
    - Mapper
      - Get per-node PageRank value as input
      - Evenly distribute PageRank value of each node to its successors



Three Main Modules (Cont.)

- Main loop in PageRank.java (for each iteration)
  - MapReduce job to update PageRank value without random jump
    - Reducer: recover node structure and update PageRank value
    - **NOTE**: the value type of reducer is node structure (PRNodeWritable.java)
      - Provide not only **full information** of node m, but also **PageRank value** given by each predecessor ➔ **carefully design**! (also in part 1: not only **information** but also **distances**)



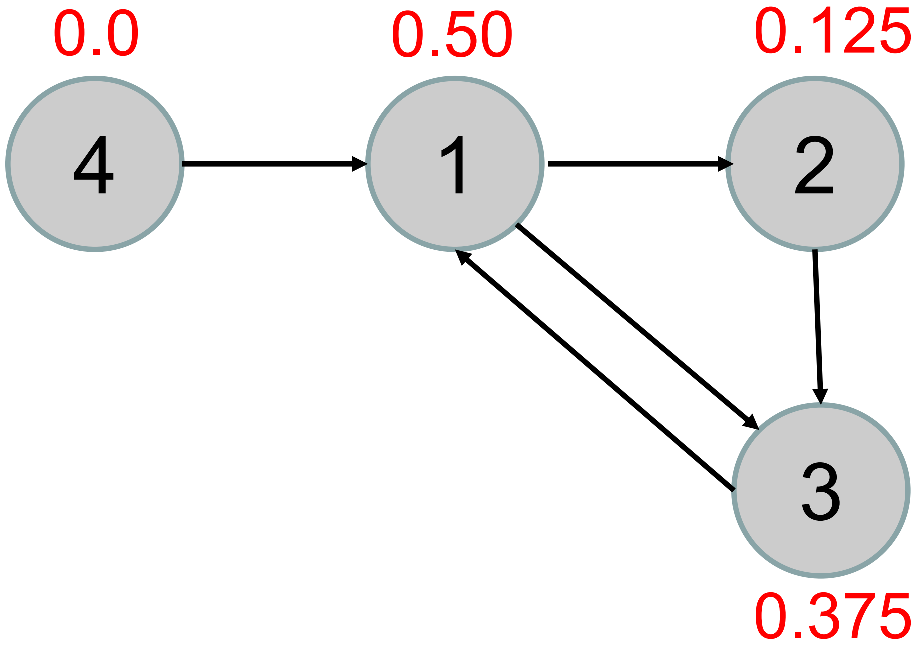
```
class Reducer
    method Reduce(nid $m$, $[p_1, p_2, \ldots]$)
        $M \leftarrow \emptyset$
        for all $p \in$ counts $[p_1, p_2, \ldots]$ do
            if IsNode($p$) then
                $M \leftarrow p$                                  ▷ Recover graph structure
            else
                $s \leftarrow s + p$                              ▷ Sum incoming PageRank contributions
        $M$.PageRank$\leftarrow s$
        Emit(nid $m$, node $M$)
```

Node structure of node m

New PageRank value of node m got from predecessors

Three Main Modules (Cont.)

- Main loop in PageRank.java (for each iteration)
  - MapReduce job to update PageRank value without random jump
    - Reducer: recover node structure and update PageRank value



Three Main Modules (Cont.)

- Main loop in PageRank.java (for each iteration)
  - We do not consider the dangling nodes in this assignment.
    - There is no dangling node in the large dataset.

Outline

- Problem
  - Measure web page quality by PageRank algorithm
- Three main modules
  - PRPreProcess.java
  - PRNodeWritable.java
  - PageRank.java
- Implementation hints
- Submission

Implementation Hints

- ➢ Datatype of PageRank value
  - For example, for a graph of six nodes, each node will have initial PageRank value of 1/6 (0.166…)
  - For accuracy, you should use **double** when calculating PageRank value
- ➢ Stop condition
  - Stop after a given number of iterations
- ➢ Cleanup
  - Remember to transform your result to required output format at last, otherwise you might **lose** grade

Implementation Hints (Cont.)

- ➢ How to chain multiple iterations?
  - The output in iteration(k) serves as the input in iteration(k+1)
- ➢ How to obtain the total number of nodes?
  - See Counter API and Example
- ➢ How to pass parameters for different jobs?
  - Set parameter in job configuration

Outline

- ➢ Problem
  - Measure web page quality by PageRank algorithm
- ➢ Three main modules
  - PRPreProcess.java
  - PRNodeWritable.java
  - PageRank.java
- ➢ Implementation hints
- ➢ Submission

Submission

- Submit at least the following files (Additional files allowed)
  - Part 2
    - ParallelDijkstra.java
    - PDNodeWritable.java
    - PDPreProcess.java
  - Part 3
    - PageRank.java
    - PRNodeWritable.java
    - PRPreProcess.java
  - **Group declaration form**
    - http://www.cuhk.edu.hk/policy/academichonesty/Eng_htm_files_(2013-14)/p10.htm

Submission (Cont.)

- Submit at least the following files
  - Follow the submission instructions under the Assignments section.
  - You can enter your source code directory, and get the tarball by
    - **tar cvzf asgn2-{SID}.tar.gz *.java {your_declaration_form}**
  - **Double-check** your tarball to ensure that it includes all required files
    - Copy asgn2-*SID*.tar.gz to a temporary directory
    - Extract it for double-check: **tar xvzf asgn2-{SID}.tar.gz**

Thank You
Q&A