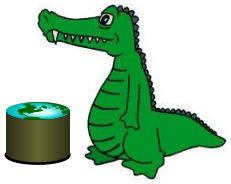
CAP4770/5771
Lab 3
PageRank implementation

University of Florida, CISE Department
TA: Xiaofeng Zhou

Hadoop version

- Your code will be tested under EMR AMI version 3.9 with Hadoop version 2.4.0.
- You need to use Java for Lab 3.
- We suggest you use Eclipse to develop and debug (and test run on small dataset) locally.
  (You don't have to setup Hadoop on your own machine, just include necessary Hadoop library in your Eclipse project setup.)



Input & output

Your program should take two parameters:

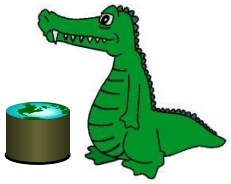
1. Input: path-to-xml-file
2. output: your-output-root-dir(which includes)

   results/PageRank.outlink.out
   results/PageRank.n.out
   results/PageRank.iter1.out (output file for itera.on 1)
   results/PageRank.iter8.out (output file for itera.on 8)
   job/PageRank.jar (your job jar)
   tmp/ (temporary files, you might or might not need it)
   logs/ (the job log directory, optional)

Input & output - cont

And make sure:

3. Use HDFS API (org.apache.hadoop.fs. FileSystem) to handle input & output. (works for local FS and HDFS)



PageRank Driver Function

```
int main(String[] args){
//job 1 extract wiki and remove redlinks
PageRank.parseXml("wiki/data","wiki/ranking/iter0---raw")
//job 2 wiki adjacency graph generation
PageRank.getAdjacencyGraph("wiki/ranking/iter0---raw",
"wiki/ranking/iter0")
//job 3 total number of pages
PageRank.calTotalPages("wiki/ranking/iter0","wiki/ranking/N")
//job 4: iterative MapReduce
for(int run =0;run<8; run++)      {
PageRank.calPageRank("wiki/ranking/iter"+String(run)","wiki/ranking/iter"
+String(run+1)) }
//job 5: Rank page in the descending order of PageRank
PageRank.orderRank() } //end of main()
//Re-organise output directory if necessary
```

Job1: extract links and remove red links

Extract links

1) Use XmlInputFormat

Mahout's XmlInputFormat will process XML files and extract out the XML

```
<main>
  <person>
   <name>Bob</name> <dob>1970/01/01</dob>
  </person>
</main>
```

and configure the start / end tags to be <person> and </person>, then your mapper will be passed the following <LongWritable, Text> pair to its map method:

LongWritable: 10 Text: "<person>\n <name>Bob</name>\n <dob>1970/01/01</dob>\n </person>”

Job1: extract links and remove red links - cont

Extract titles and links

1. Title A can be simply extracted between <title> A </title>. No complex rule is needed to extract A. Just take what it is between <title> and <\title>.
2. Extract the wikilinks.

Job1: extract links and remove red links - cont

Example:

<page>

<title> **AccessibleComputing**</title> ------ extract AccessibleComputing for simplicity.

<redirect title = “Computer accessibility”> ------ignore the redirect title

<text> [[**Computer accessibility**]] ---- extract the link

</page>

<page> <title> **Anarchism** </title> ------extract Anarchism

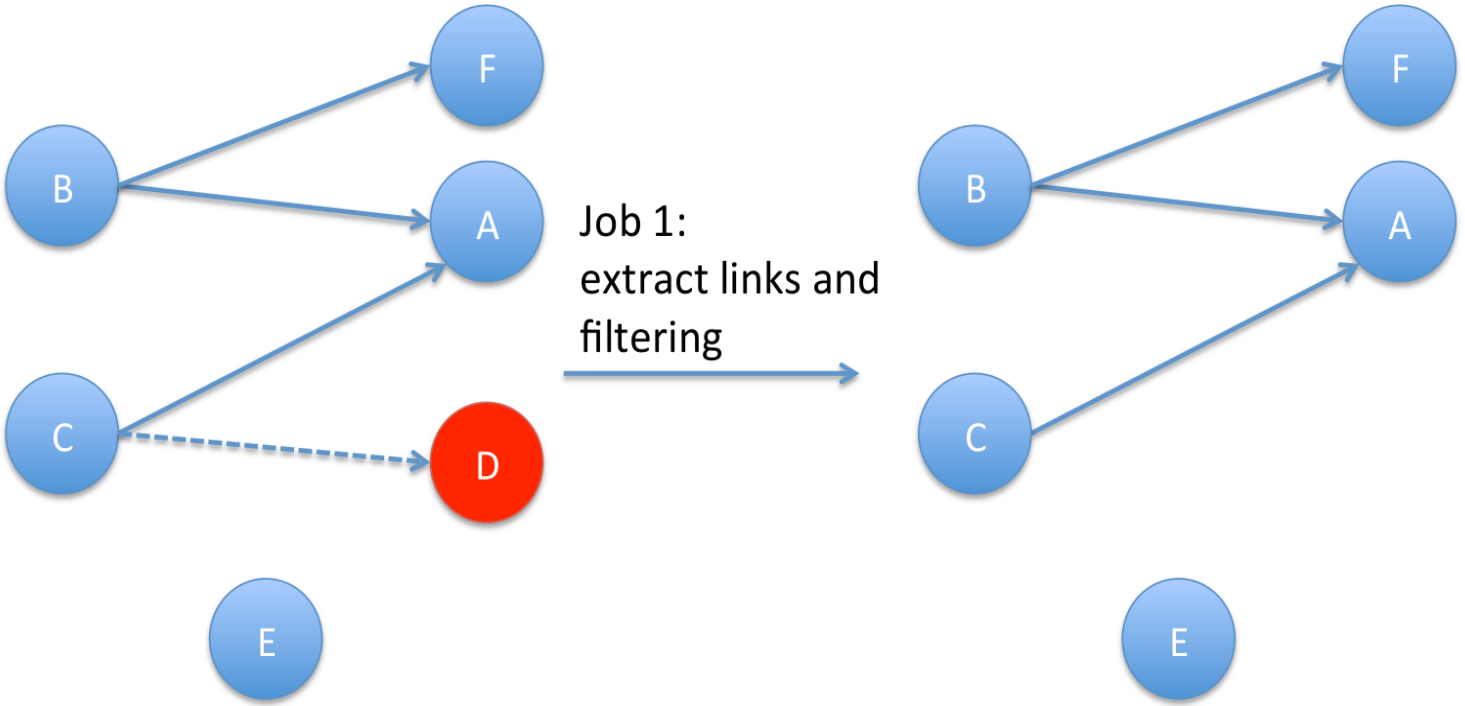
<text> .... Is a [[**political philosophy**]] that advocates [[**stateless society** | stateless societies]] of defined as [[**self---goverance**|self--- governed]]....

Job1: extract links and remove red links - cont

For wikilink:

1. Case sensitive
2. Replace empty space in title and wikilink with ‘_’.
3. No other processing steps needed

Job1: extract links and remove red links - cont



D needs to be removed since there is no page D in the wiki dataset although page C mentions link D in its page(hint:create ingraph and filter out redlink)

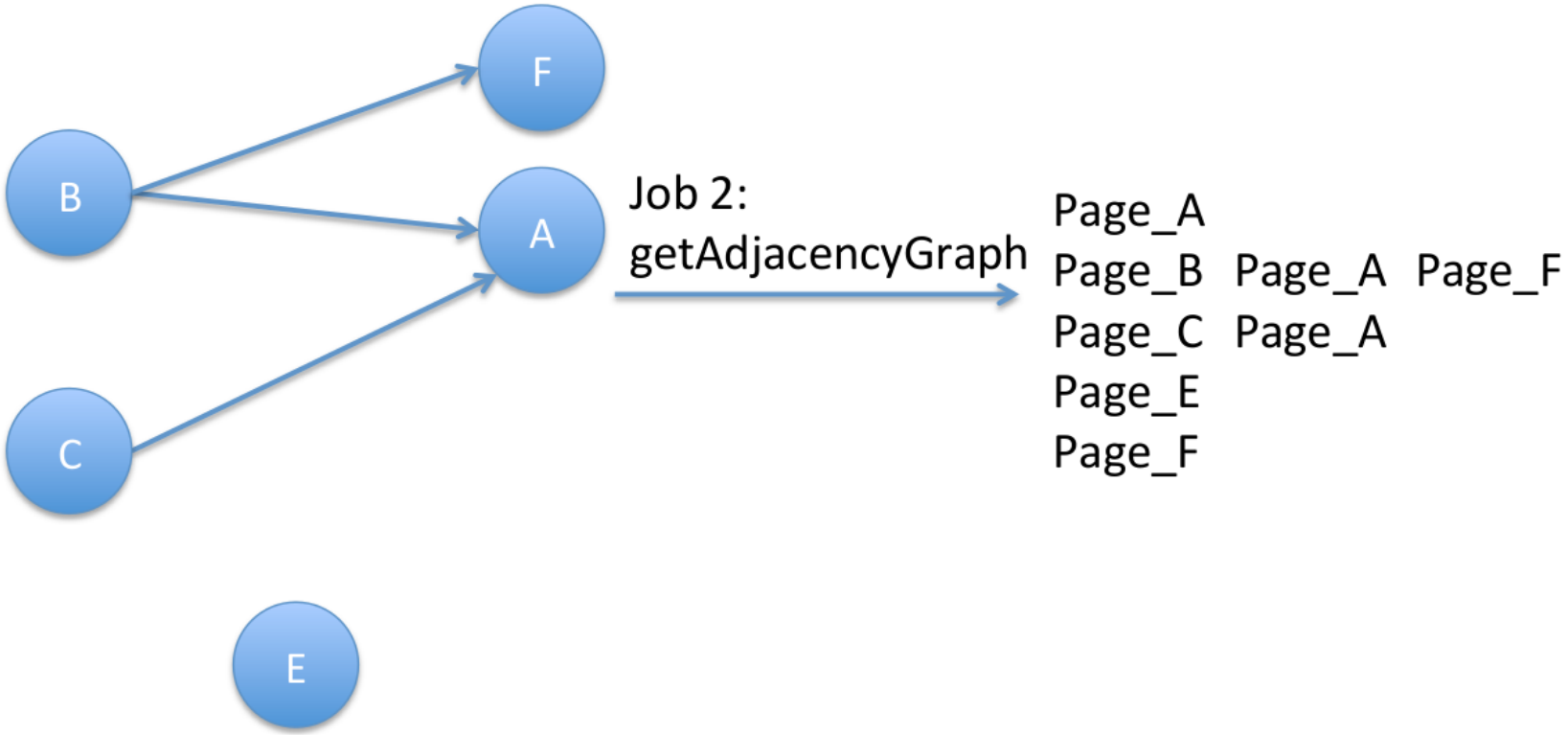
Job2: adjacency graph (outgraph)

The output of Job2 should be a adjacency graph. The adjacency graph format is:

<src> <links in page>

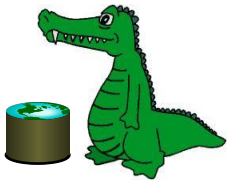
1. The <src> is the title of the page.
2. The <links in page> is the list of wikilinks found in page <src>.
3. The <links in page> should not contain red links
4. The <links in page> should not contain duplicate links and it should not contain a link which points to the page itself.
5. All the data is separated by a tab.

Job2: adjacency graph (outgraph)



Job3: N calcuation

N is simply the number of <title>...<\title> pairs discovered in the dataset.(or line count of the adjacency graph(outgraph))



Job4: PageRank Calculation

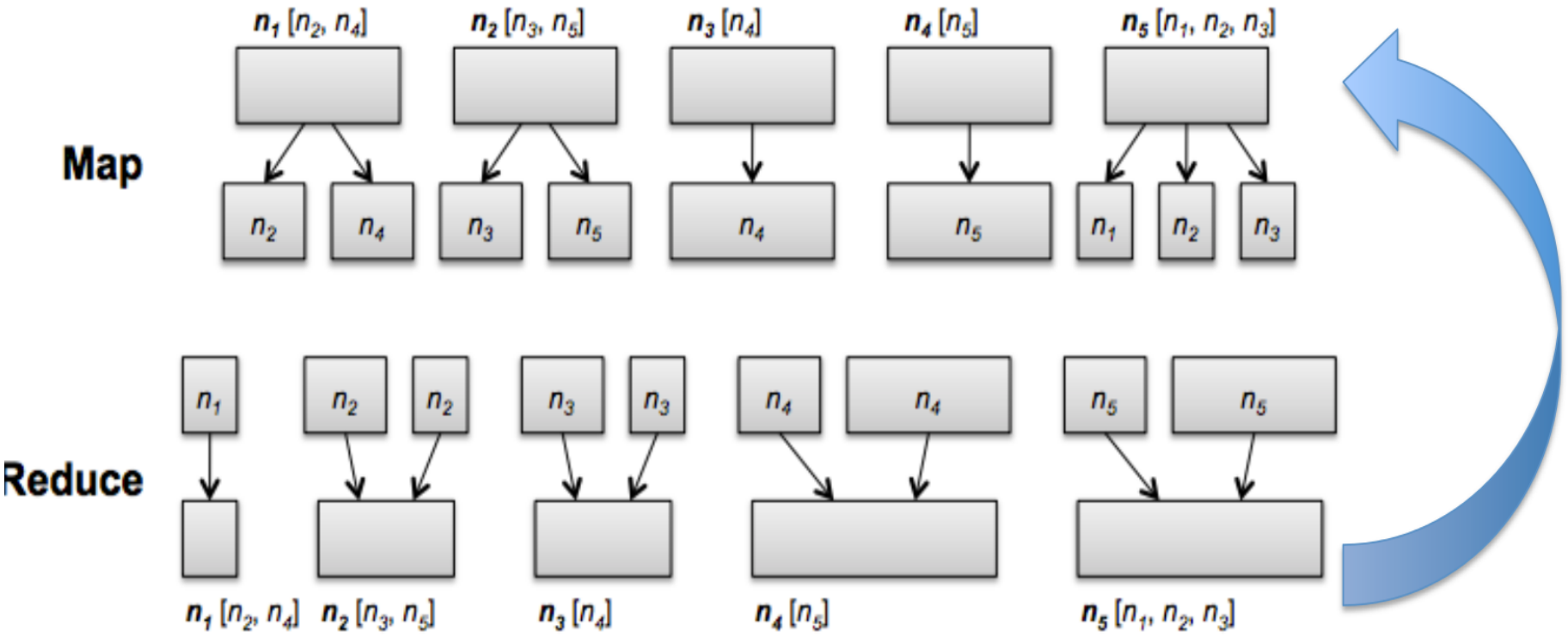
Algorithm 5.3 PageRank (simplified)

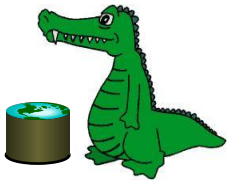
In the map phase we evenly divide up each node's PageRank mass and pass each piece along outgoing edges to neighbors. In the reduce phase PageRank contributions are summed up at each destination node. Each MapReduce job corresponds to one iteration of the algorithm. This algorithm does not handle dangling nodes and the random jump factor.

```
1: class Mapper
2:     method Map(nid n, node N)
3:         p ← N.PageRank/|N.AdjacencyList|
4:         Emit(nid n, N)                          ▷ Pass along graph structure
5:         for all nodeid m ∈ N.AdjacencyList do
6:             Emit(nid m, p)                      ▷ Pass PageRank mass to neighbors

1: class Reducer
2:     method Reduce(nid m, [p1, p2, ...])
3:         M ← ∅
4:         for all p ∈ counts [p1, p2, ...] do
5:             if IsNode(p) then
6:                 M ← p                           ▷ Recover graph structure
7:             else
8:                 s ← s + p                       ▷ Sum incoming PageRank contributions
9:         M.PageRank ← s
10:        Emit(nid m, node M)
```

Job4: PageRank Calculation - cont

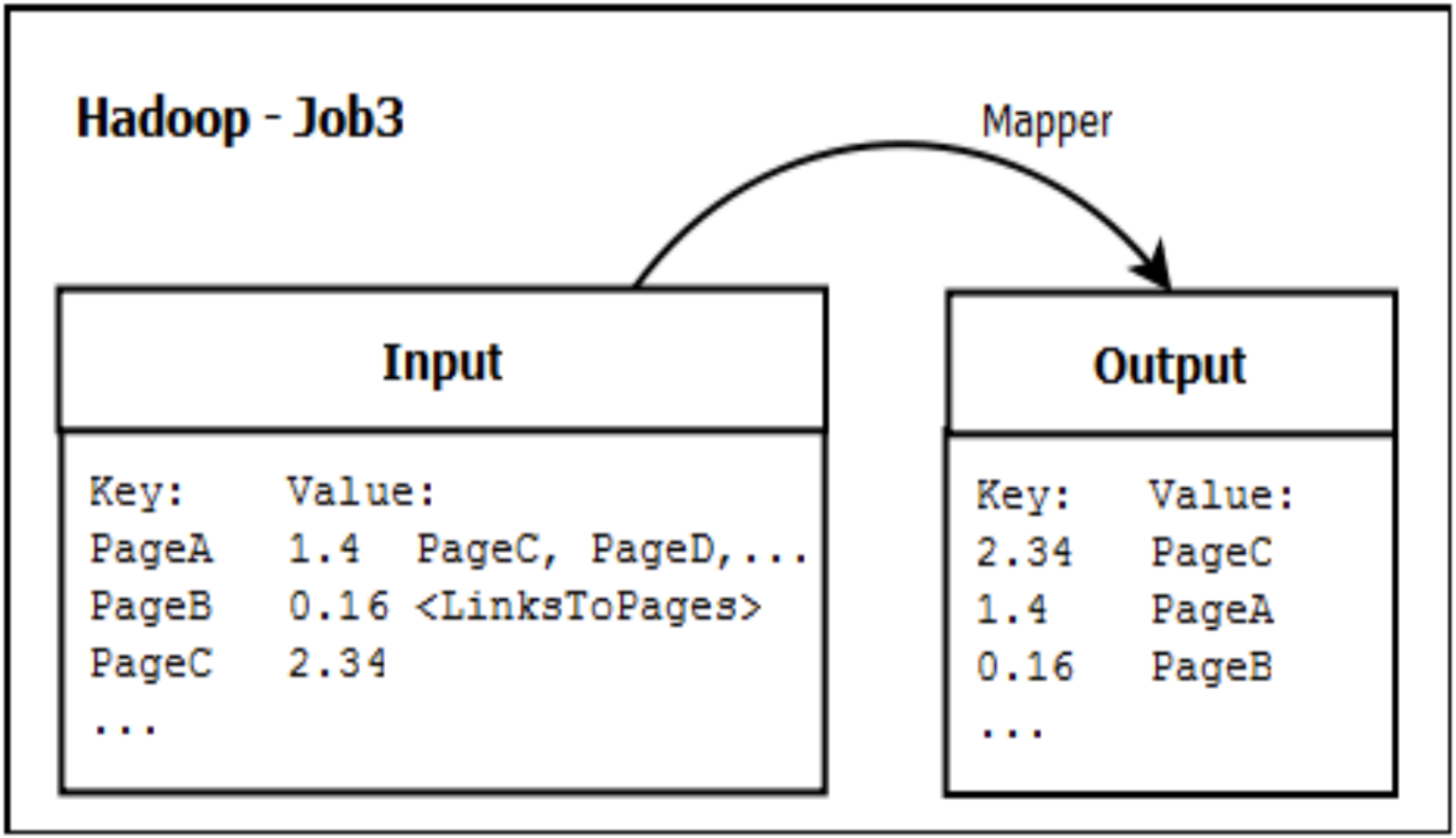


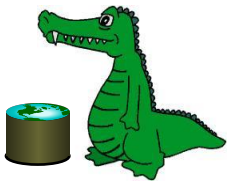


Job 5: PageRank Ordering

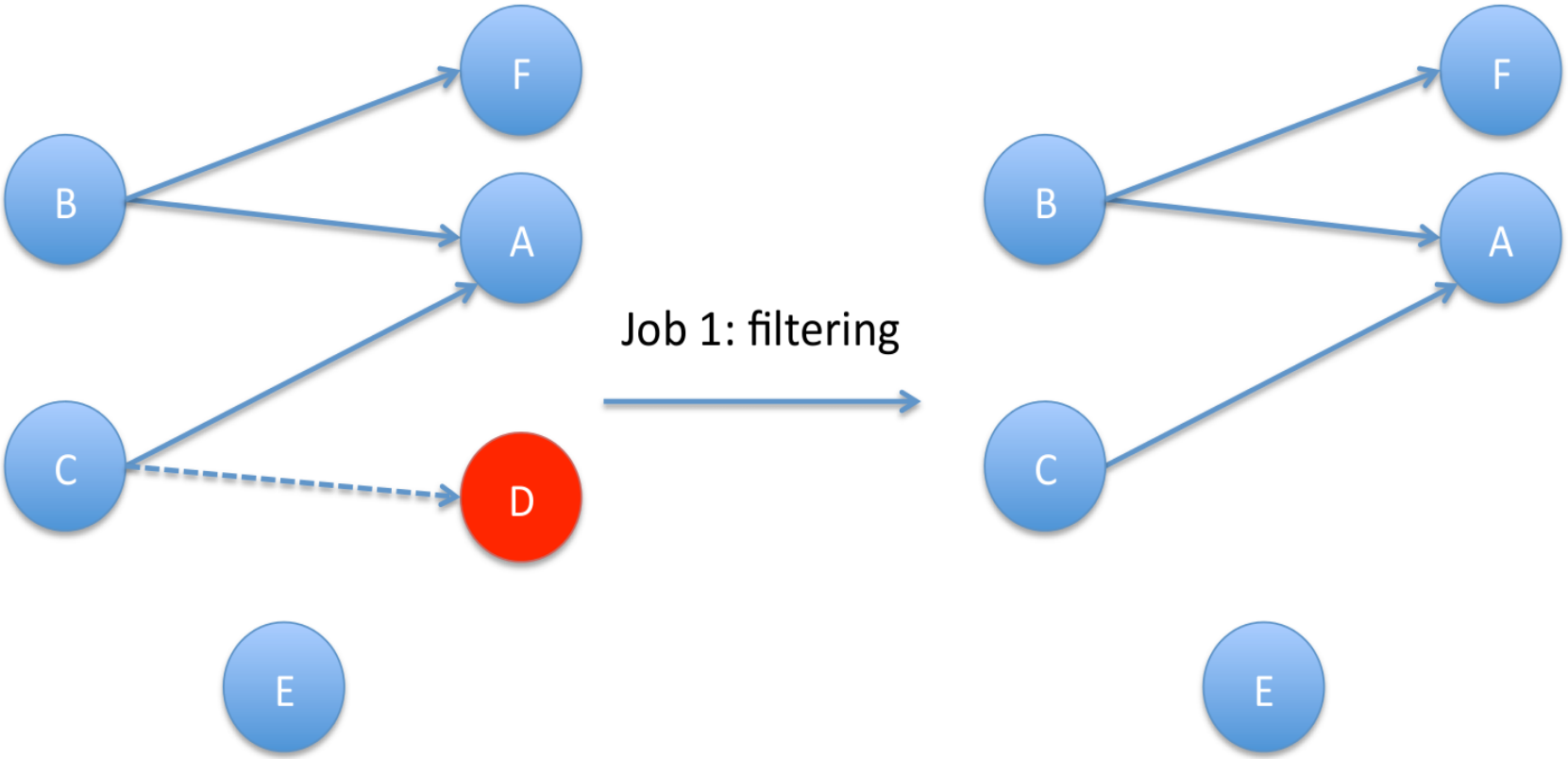
1. Filtering: only print out the page with PageRank >= 5/N (in the Map function)
2. Emit (PageRank, Page)
3. Only one Reducer
4. Output the result in the descending order of PageRank. Here you will need to override the default sorter to sort in decreasing order(extends WritableComparator).

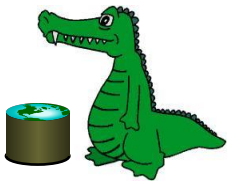
Job 5: PageRank Ordering - cont



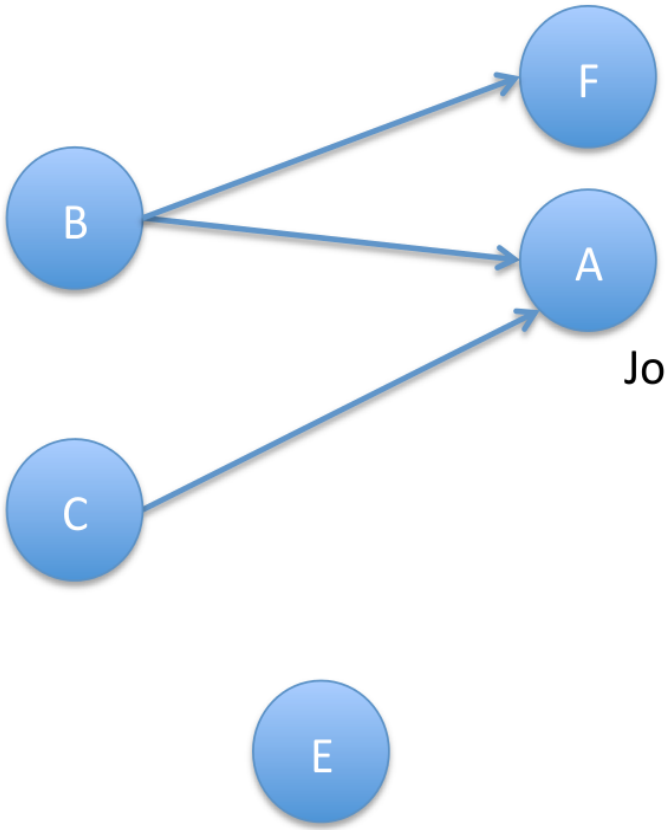


Example Job 1: Extract wikilinks and remove red links





Example Job 2: adjacency graph

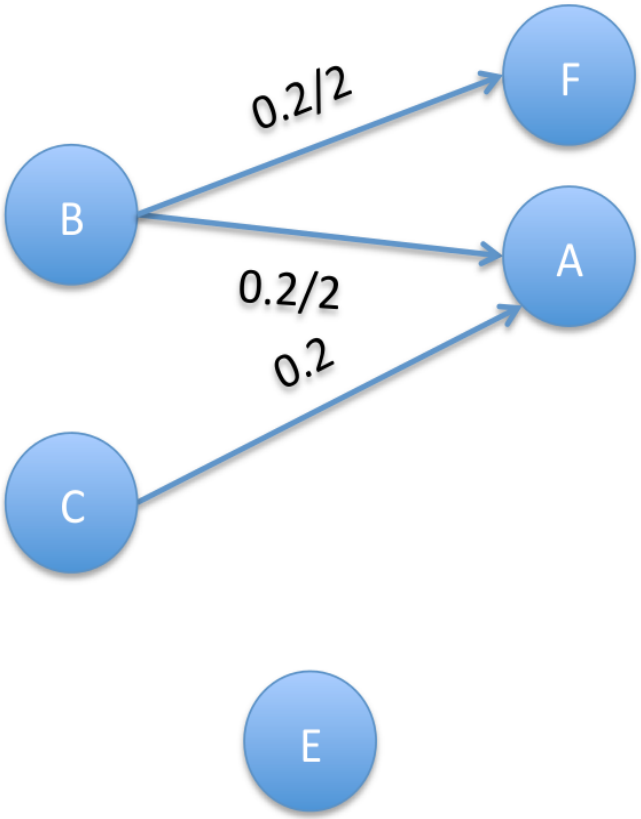


Job 2: adjacency graph

Adjacency graph (PageRank.outlink.out)

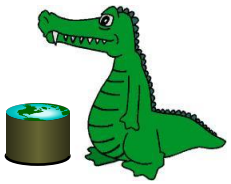
B A F
A
F
C A
E

Examples: Iteration 1

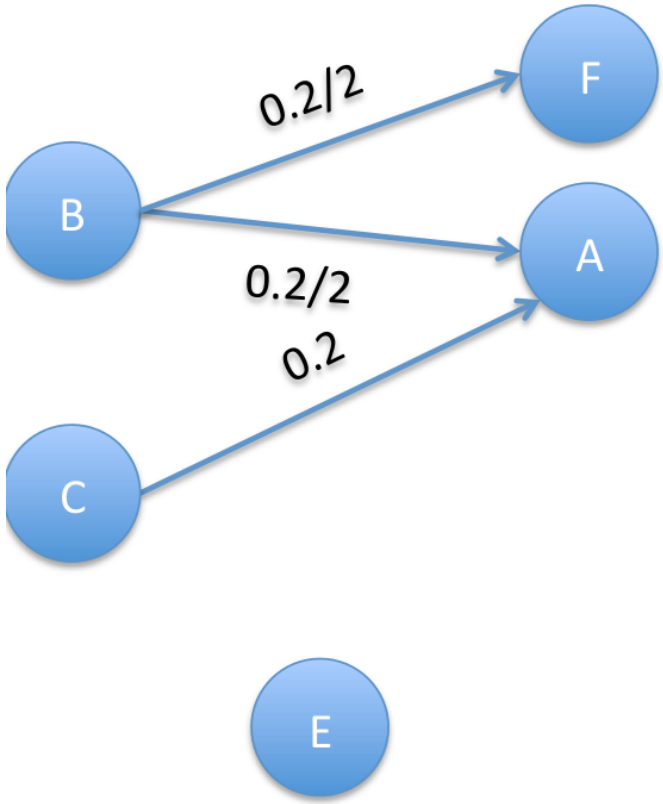


1) Total number of pages N=5. It contains A,B,C,E,F
2) E is a standalone page. It has no links. But E and its PageRank need to be printed out.

P(A) = (1-0.85)/5 + 0.85*(0.2/2 + 0.2/1) = 0.285
P(F) = (1-0.85)/5 + 0.85*(0.2/2) = 0.115
P(B) = (1-0.85)/5 = 0.03
P(C) = (1-0.85)/5 = 0.03
P(E) = (1-0.85)/5 = 0.03



Examples: Iteration 2

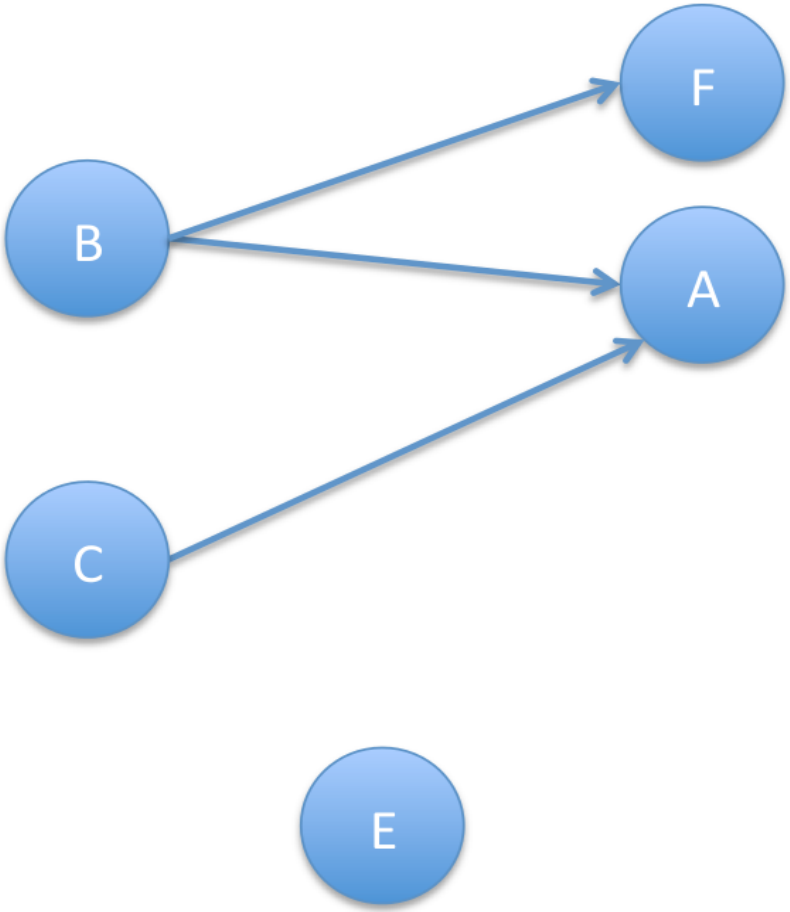


1) Total number of pages N=5. It contains A,B,C,E,F
2) E is a standalone page. It has no links. But E and its PageRank need to be printed out.

P(A) = (1-0.85)/5 + 0.85*(0.03/2 + 0.03/1) = 0.06826
P(F) = (1-0.85)/5 + 0.85*(0.03/2) = 0.04275
P(B) = (1-0.85)/5 = 0.03
P(C) = (1-0.85)/5 = 0.03
P(E) = (1-0.85)/5 = 0.03

In our project, we don't use teleport to deal with the sink node for simplicity. At the ini.al point, the sum of the PageRank is 1. But the sum will gradually decrease with the iterations due to the PageRank leaking in the sink nodes.

Example: Results



PageRank.n.out

N=5

PageRank.outlink.out

Page_A

Page_B Page_A Page_F

Page_C Page_A

Page_E

Page_F

PageRank.iter1.out

Page_A 0.285

Page_B 0.03

Page_C 0.03

Page_E 0.03

Page_F 0.115