Hadoop version

- Your code will be tested under EMR AMI version 2.4.2
- You can develop and test your code using Hadoop 1.0.3, which is corresponding to the AWS EMR AMI version 2.4.2
- http://archive.apache.org/dist/hadoop/common/hadoop-1.0.3/
- You may have some incompatible issue using other Hadoop versions.

Development process

① First install Hadoop in your local computer and test your code with a smaller dataset.

② Then migrate to AWS EMR and test your code with the large dataset.

If you have problem with environment configuration, you can also use Amazon EMR as a build environment to compile programs for use in your cluster.

http://docs.aws.amazon.com/ElasticMapReduce/latest/DeveloperGuide/emr-build-binaries.html

PageRank Driver function

```
int main(String[] args) {
  //  job 1  extract wiki and remove red links
   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()
  }
```

Job1: extract links and remove red links

Extract links

1) Use XmlInputFormat

Mahout's XmlInputFormat will process XML files and extract out the XML between two configured start / end tags. So if your XML looks like the following:

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

and you've configured 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

Extract title 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. We are no longer suffering to make a complex regular expression to extract the wikilinks since the red links will be thrown away in this job.

Job1: extract links and remove red links

Extract title and wikilink in the page

<page>

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

<redirect title = "Computer accessibility"> --ignore the redirect title

<text> [[Computer accessibility]]

</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

Extract wikilink

① Assume case sensitive.

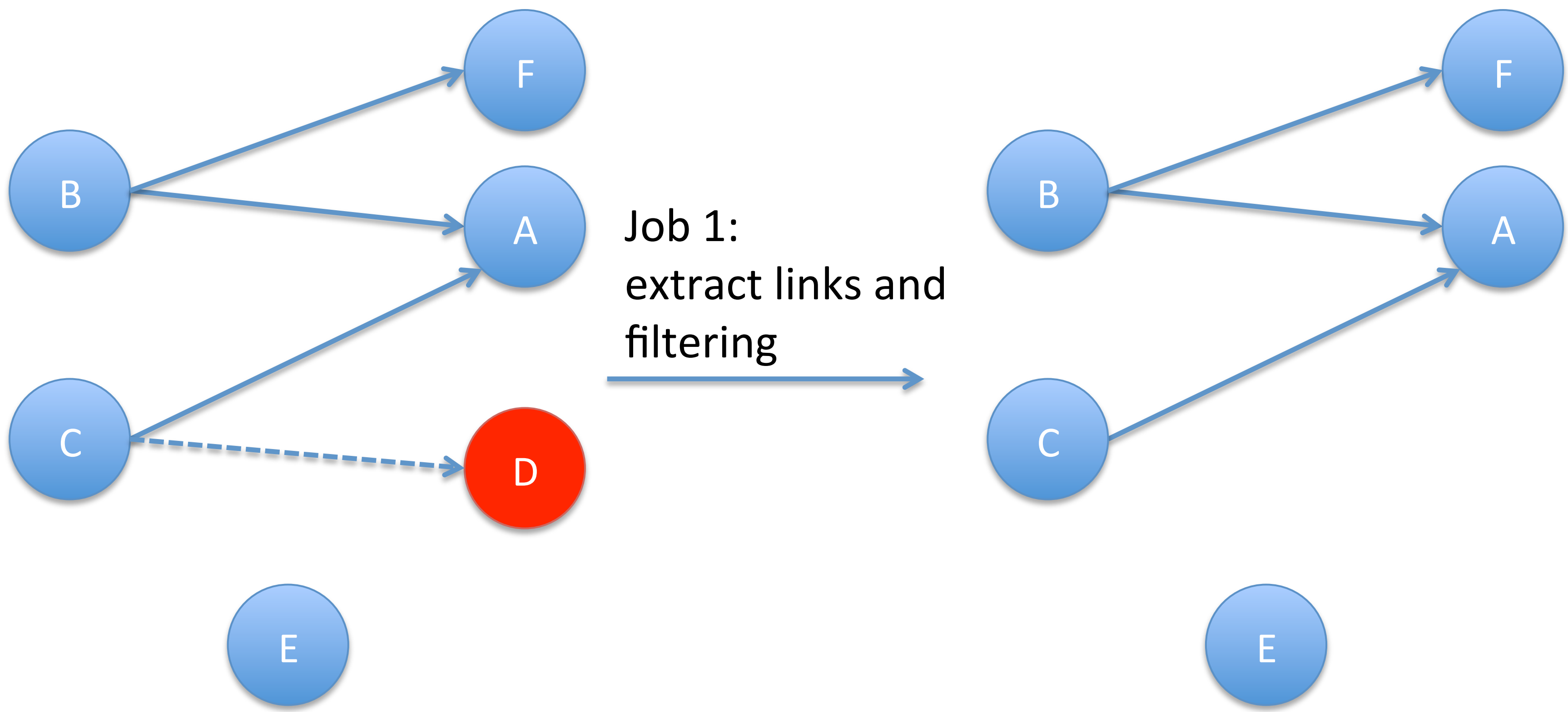
② No other sophisticated processing is needed.

③ Replace empty space in title and wikilink with ‘_’.

Job1: remove red links

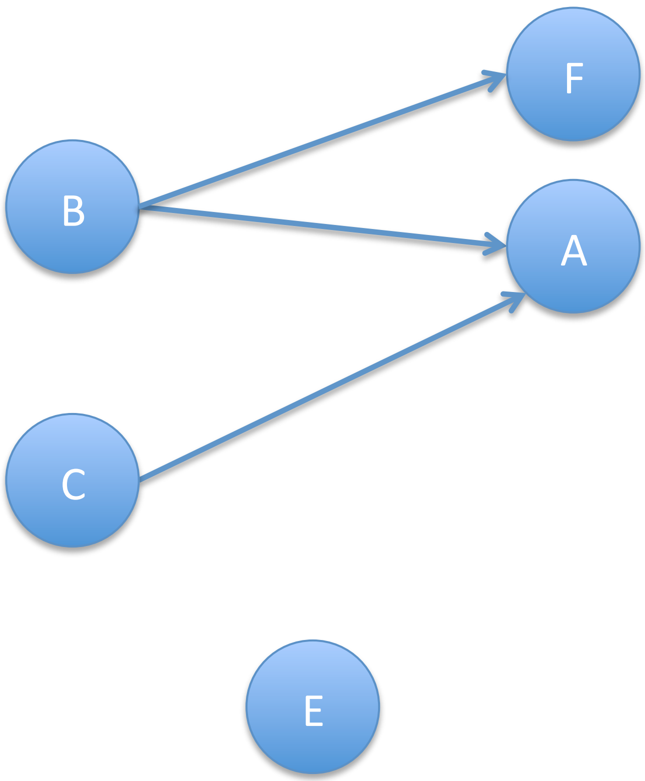
- What is red links? http://en.wikipedia.org/wiki/Wikipedia:Red_link
- Let's remove all the red links first before the PageRank calculation.
- How to remove red links in the project? Write a MapReduce job to throw away the red links.

Job1: extract wiklinks and remove red links



D needs to be removed since there is no page D in the wiki dataset although page C mentions link D in its page. The nonexistence of page D simply means we are not able to find a <title>D<\title> in the dataset.

Job2: adjacency graph



Job 2:
getAdjacencyGraph

Page_A
Page_B Page_A Page_F
Page_C Page_A
Page_E
Page_F

Job2: adjacency graph

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.

Job3: N calcuation

N is simply the number of <title>…<\title> pairs discovered in the dataset.

Job4: PageRank Calculation

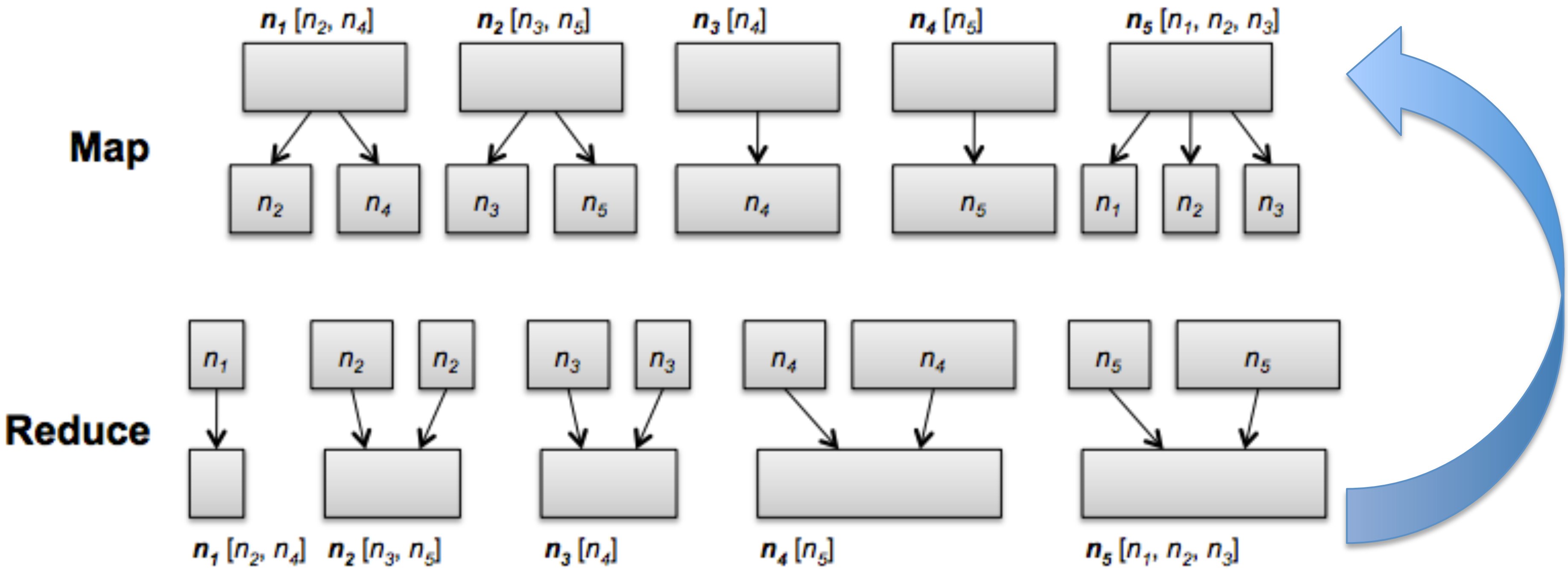
Algorithm 5.3 PageRank (simplified)

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.

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

```
class Reducer
    method Reduce(nid m, [p1, p2, ...])
        M ← ∅
        for all p ∈ counts [p1, p2, ...] do
            if IsNode(p) then
                M ← p                                   ▷ Recover graph structure
            else
                s ← s + p                               ▷ Sum incoming PageRank contributions
        M.PageRank ← s
        Emit(nid m, node M)
```

Job4: PageRank Calculation



Job 5: PageRank Ordering

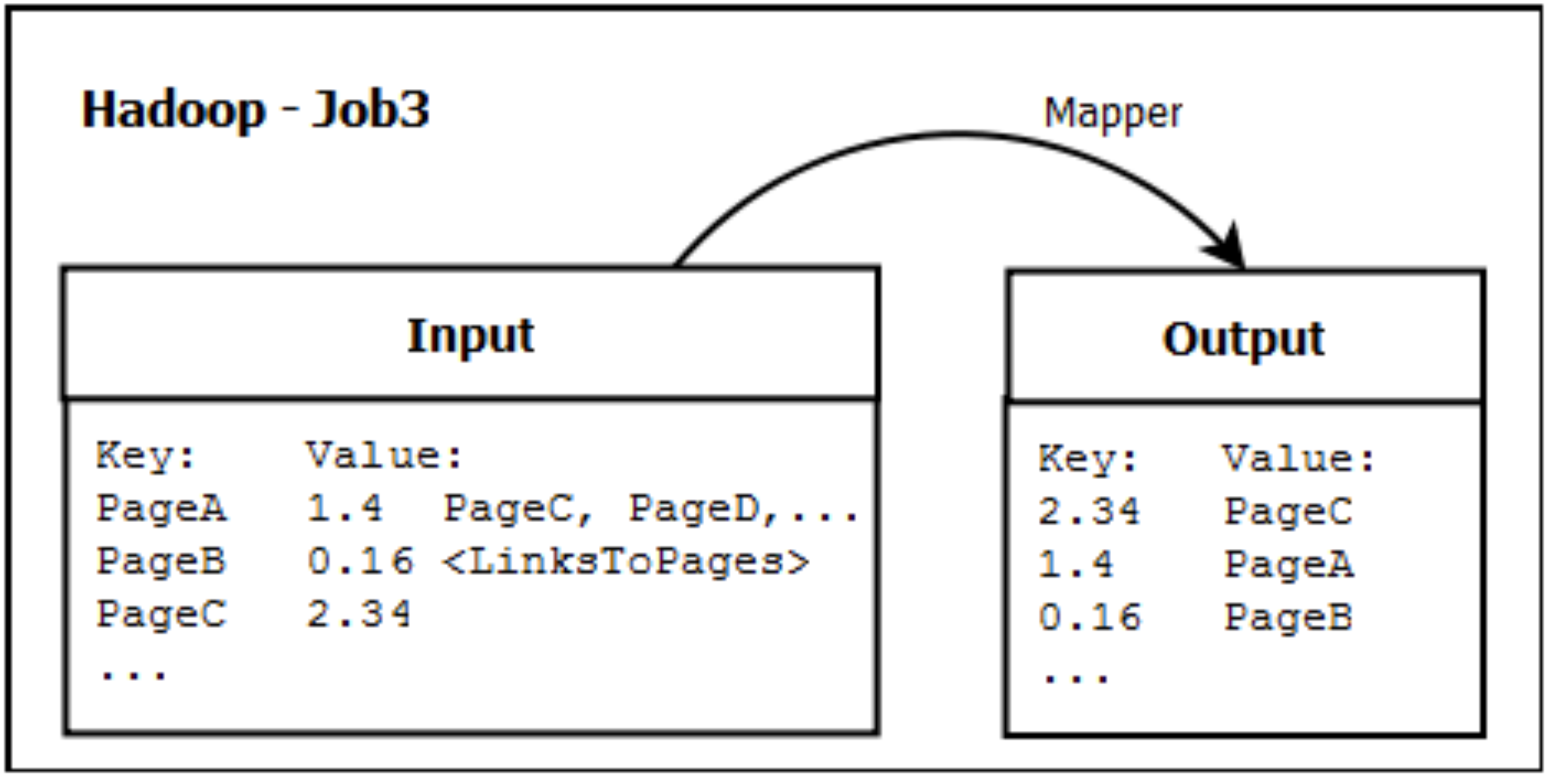
1. Filtering: only print out the page with PageRank >= 5/N (in the Map function)

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



Project inputs & outputs

- Input

spring-2014-ds/**data**/enwiki-latest-pages-articles.xml

your-bucket-name

results/PageRank.outlink.out

results/PageRank.n.out

results/PageRank.iter1.out (output file **for** iteration 1)

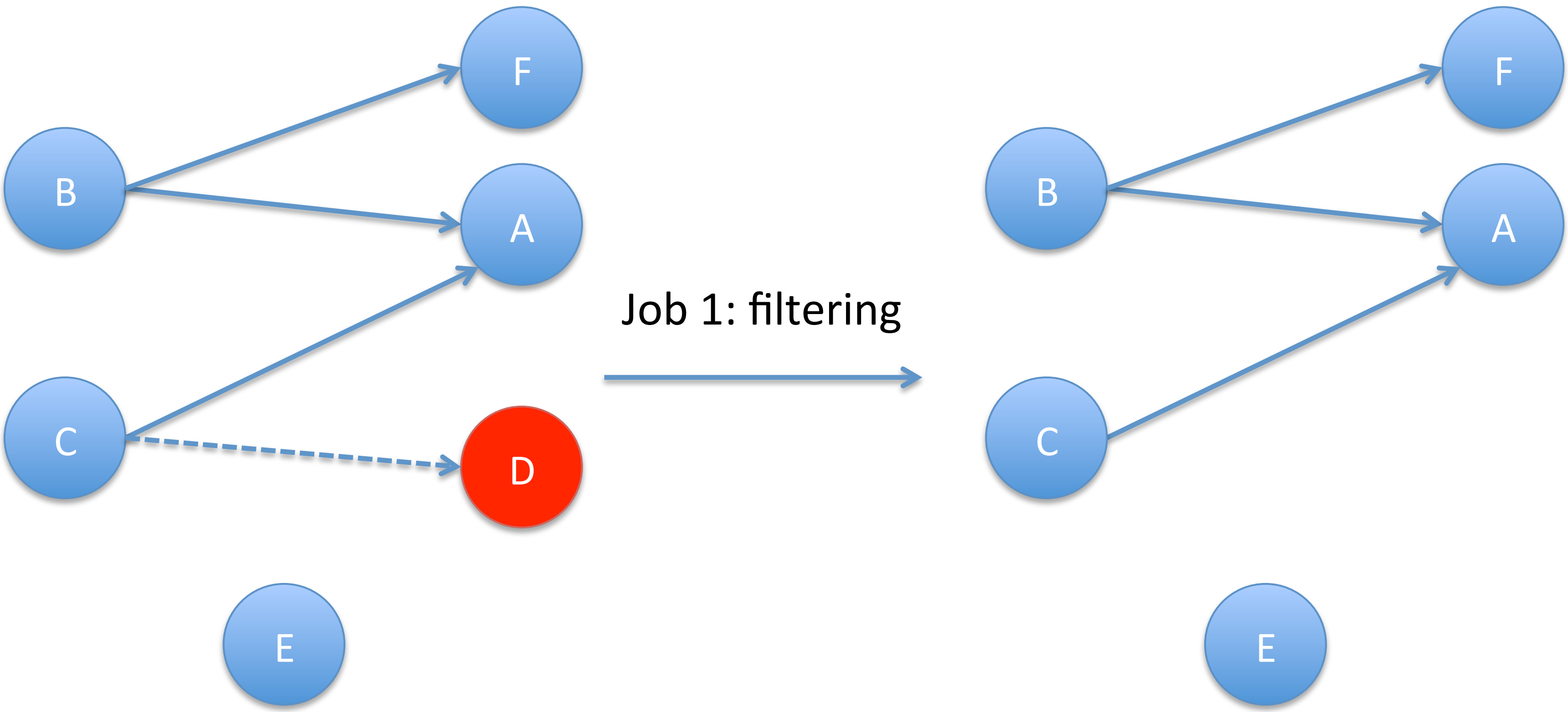
results/PageRank.iter8.out (output file **for** iteration 8)

logs/ (the **job** log directory)

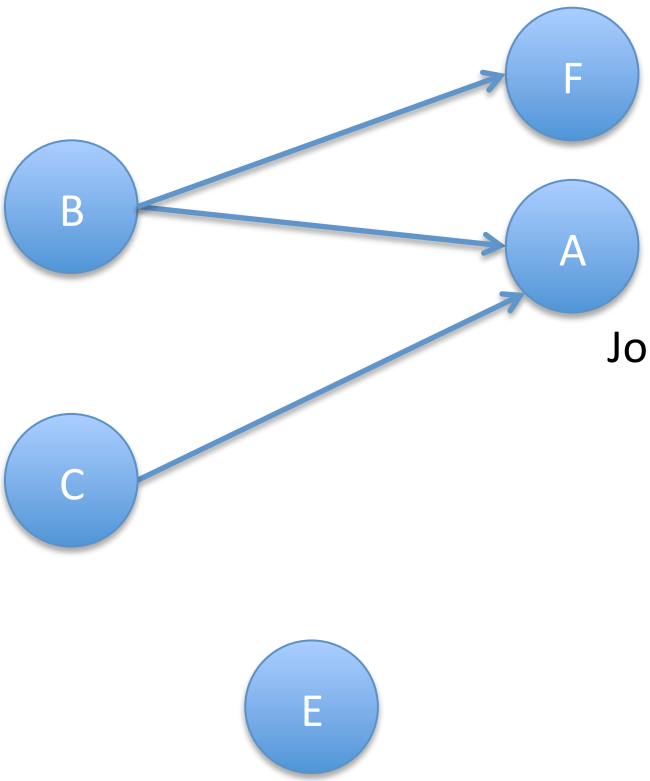
job/PageRank.jar (your **job** jar)

tmp/ (temporary files, you might or might not need it)

Example Job 1: remove red links



Example Job 2: adjacency graph

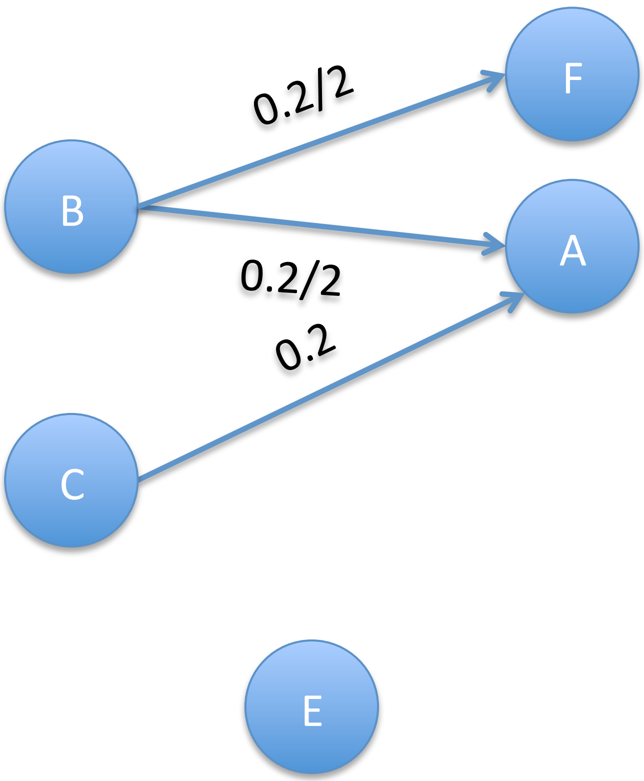


Job 2: adjacency graph

Adjacency graph (PageRank.outlink.out)

B	A	F
A		
F		
C	A	
E		

Examples: Iteration 1

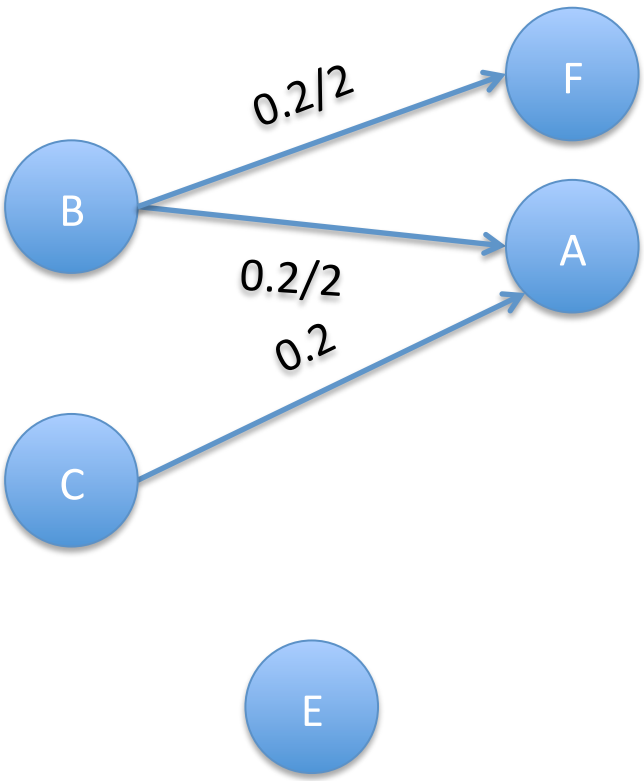


1) Total number of pages N=5. It contains A,B,C,E,F
2) E is a standalone page. It has no links. But E and its PageRank need to be printed out.

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

In our project, we don't use teleport to deal with the sink node for simplicity. At the initial 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.

Examples: Iteration 2

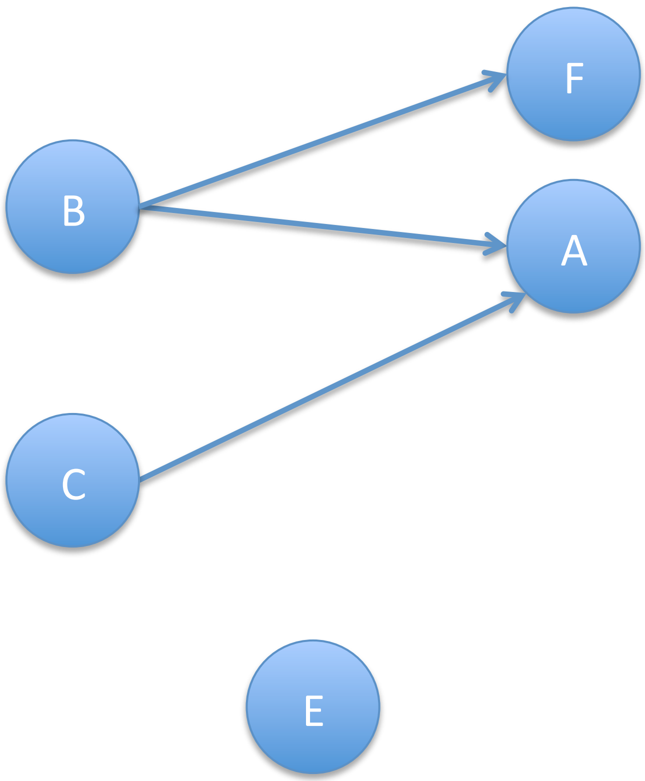


1) Total number of pages N=5. It contains A,B,C,E,F
2) E is a standalone page. It has no links. But E and its PageRank need to be printed out.

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 initial 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

What is the outlink? Answer

Questions?