Name: Chandrika Sharma

GITHUB: https://github.ccs.neu.edu/chandrika2311/MapReduce-CS-6240

I have made a new class in assignment 3 the git-repo called

1. MatrixPageRank.java
2. MatrixColumnPageRank.java

PSEUDOCODE:

Parallel Matrix Multiplication: Row by column Partition Approach:

JOB1: Preprocessing

Preprocessing Mapper

```
map(Object key, Text value){

Bz2WikiParser1 bz2 = new Bz2WikiParser1();
String x = bz2.parsing(String.valueOf(value));
For all link in outlinks:
        Emit(page Name, link)
}
```

Preprocessing Reducer

```
reduce(Text key, ListOfOutlinks){

        emit(key, SetOfOutlinks)

}
```

JOB2: Creating the file with keys of the matrix and R matrix: Set Reducer To 1

Upload the graph from Job1 to Mapper, then emit all the (1, link)
Multiple outputs used here to emit the keys file and to make the R matrix.

```
Map(map(Object key, Text value){
Emit(1,link+":"+out-link-count)
}
```

On receiving all the links with key 1, emit each link with a counter value. Since counter cannot be maintained globally, setting number of reducers to 1

```
Reducer (1, [links]) {
Count = 0;
Graph Size = x
Initial PageRank = 1/ Graph Size
For link in links:
        Emit (link, count+":"+out-link-count)
        Emit(count, link, Initial PageRank)

}
```

Now the output key file and R-Matrix looks like following:

KeyFile:

Link	id:outlinkCount
MCLinux_355e	8:0
They_Might	2016:3
Theism	2037:7
Theia	2038:8
Theft	2039:8

RMatrix:

500 Wrench:5.257623554153522E-5
501 Wrapping_paper:5.257623554153522E-5
502 Worms,_Germany_b18a:5.257623554153522E-5
503 Worms:5.257623554153522E-5

JOB3: Here we create the matrix with the keys:

PLEASE NOTE: Using file cache, I pass the key file to all the machines so that the keys are available to each machine to assign to the new matrix.

Mapper(){
Setup(){
Put all the links from the key File into a hash map

}

map(){

iterate over the graph and assign each link with its ID from the hash map and emit by **inversion of order:**
so if the record is as follows: **o2, o3, o4 here are the number of out links**

"link1: o1 link2:o2,link3:o3,link4:o4"

and say link1 has id – 1, link2 has id – 2, link3 has id – 3 and so on.
Then:
Emit (2: o2,1: o1)
Emit (3: o3,1: o1)
Emit (4: o4,1: o1)

}

Reduce(){
Now here we create a sparse matrix with link and list of its in links.

10010:10 7940:505,10010:10,10011:5,
10011:5 9394:507,7929:382,10011:5,6295:535,
10012:17 10005:20,14731:19,7311:9,6559:154,9288:9,3709:5,
10013:13 5376:38,13115:29,3291:28,10013:13,

Here 10010:10 is a link with number of out links equal to 10

And "7940:505,10010:10,10011:5"

7940 is the in link for 10010 and it has 505 out links.
Similarly, 10011:5 and 10010:10

So our new matrix is inverted in the sense that, now instead of "link: list of out links", it depicts "link: list of in links" with each link's count of out links concatenated with semicolon.
}

JOB4: This JOB is to do the page rank calculations.

Mapper(){

Map(){
Take each record from the new matrix:
Emit:(row_id,row_col_id_value)
Example: 10010:10 7940:505,10010:10,10011:5

Emit, (10010,"10010,7940=505")
And (10010,"10010, 10010=10")
And (10010,"10010, 10011=5")

Here, 505,10 and 5 are the cj values. Row id is 10010 and column id = 7940,10010,10011

}

reducer(){
setup(){

1. **Load the R matrix from the file cache and make a hash map with it. Key will be the key of the link. And page name also is included in this as R matrix has it.**

}
reduce(){
get old page rank value form the hash map made in setup,
get the total number of out links(cj) in the record as sent by mapper.
For each values:
Running total = Summation of 0.85∗(1/cj)∗PR_from_hashmap;

}
emit(row_Id , page name +":"+running_total)// This is exactly how out R matrix looks like.

}

JOB5: Top 100

this will be repeated 10 times and the top 100 records with the maximum page rank will be gathered in this job

Parallel Matrix Multiplication: Column by Row Partition Approach:

The first 3 jobs are same here.
From the 4th job the intuition is as follows:

JOB4: This JOB is to do the page rank calculations.

PLEASE NOTE: Using file cache, I pass the key file to all the machines so that the keys are available to each machine to assign to the new matrix.
Mapper(){

Setup(){
Take the R matrix and create a HashMap with it.
rMatrix.put(colId,value);
here colId is the key and value is the page rank value

}

Map(){
Take each record from the new matrix:
Emit:(row_id,row_col_id_value)
Example: 10010:10 7940:505,10010:10,10011:5

Emit, (7940,"10010,7940,505")
And (10010,"10010, 10010,10")
And (10011,"10010, 10011,5")
Here, 505,10 and 5 are the cj values. Row id is 10010 and column id = 7940,10010,10011

I have emitted the column id as key so that they get partitioned based on the column numbers.

Also check if the column exists in the hashmap rMatrix and get the Beta value from there and multiply it with the cj value for that row_id = col_id

```
colValue = alpha*Beta
```

Now emit(col_id, (row_id+","+colId+","+colValue.toString()**)**

}

reducer(){

reduce(){
get all the records with same column id:
for each row:
HashMap.put(row, colValue**)**
All the rows in that column will come into this hash map by the end of the reduce phase and So the hash map will have each row and its out link counts multiplied with the page rank value on that row of the R Matrix.

}

cleanup{
In Cleanup, where all the rows are filled into the Hash Map, emit the row and the calculated value for it.
here, emit(row**, temp value)**
}

JOB5:Final Map Reduce Phase To merge values for each row

Map(){
Get all the (row,tempValue) records and emit with row as key
}
reducer(){
setup(){

1. **get R matrix into the filecache.**
2. **Make hashmap of the id,pagename:pagerank**
3. **This will be used to match the rows with the Pagenames in reduce phase**

}

}
reduce(){
get the name of the link from the hashmap by inputing the row id and add all the temp values for the row to get the file page rank.
}

TOPK: Same as assignment 3

NOT HANDLING DANGLING NODES

PERFORMANCE COMPARITION:

Below times include parsing the original input.
Dangling nodes have not been handled.

Row-Col 6 machines:

1. **6 machines: 4176 seconds**
2. **preprocessing:"launchTime":1522475976541, "finishTime":1522478321231**
3. **key and R Matrix creation:"launchTime":1522478331710,"finishTime":1522478385020**
4. **get key Matrix: "launchTime":1522478398981, "finishTime":1522478703751**
5. **do Pageranking: "launchTime":1522478716196 "finishTime":1522478851388**
6. **top100: "launchTime":1522480103556, "finishTime":1522480130786**

Preprocessing	**1522475976541**	**1522478321231**	**2344690**
key and R Matrix creation	**1522478331710**	**1522478385020**	**53310**
get key Matrix	**1522478398981**	**1522478703751**	**304770**
Do Pageranking	**1522478716196**	**1522478851388**	**135192**
top100	**1522480103556**	**1522480130786**	**27230**
Total			**2865192**

Row-Column 11 machines:

1. **time taken : 2440 seconds**
2. **preprocessing: "launchTime":1522511667824, "finishTime":1522512826140**
3. **key + R Matrix creation: "launchTime":1522512836432,"finishTime":1522512877505**
4. **get key Matrix: "launchTime":1522512891407 "finishTime":1522513062342**
5. **do Pageranking: "launchTime":1522513075805 "finishTime":1522513167678**
6. **top100 "launchTime":1522514059238 "finishTime":1522514085276**

Preprocessing	**1522511667824**	**1522512826140**	**1158316**
key and R Matrix creation	**1522512836432**	**1522512877505**	**41073**
get key Matrix	**1522512891407**	**1522513062342**	**170935**
Do Pageranking	**1522513167678**	**1522513075805**	**91873**
top100			**26038**
Total			**1488235**

Col-Row 11 machines:

1. **Time taken:** 3320 seconds
2. **Preprocessing** "launchTime": 1522531220982 - "finishTime":1522532317336
3. **create key and R-matrix:** "launchTime":1522532326849,"finishTime":1522532368152
4. **create new keys Matrix:** "launchTime":1522532382207, "finishTime":1522532556038
5. **do-ranking:** launchTime:1522532568055, finishTime:1522532828936
6. **final-ranking:** launchTime:1522532838858, finishTime:1522532904829

Preprocessing	**1522531220982**	**1522532317336**	**1096354**
create key and R-matrix	**1522532326849**	**1522532368152**	**41303**
create new keys Matrix	**1522532382207**	**1522532556038**	**173831**
do-ranking	**1522532568055**	**1522532828936**	**260881**
final-ranking	**1522532838858**	**1522532904829**	**65971**
Top100	**1522353284538658**	**1552535905829**	**26038**
Total			1664378

Col-Row 6 machines:

1. **Time taken:** 5678 seconds
2. Preprocessing "launchTime":1522535001198, "finishTime":1522537438791
3. **create key + Rmatrix:**"launchTime":1522537449545,"finishTime":1522537501793
4. **create new keys Matrix**: "launchTime":1522537515793 "finishTime":1522537777466
5. **do-ranking**:"launchTime":1522537789765 "finishTime":1522538416262
6. **final-ranking** :"launchTime":1522538426529 "finishTime":1522538492538

7. **top100**: "launchTime":1522540629736 "finishTime":15225406566

Preprocessing	**1.52254E+12**	**1.52254E+12**	**2437593**
create key and R-matrix	**1.52254E+12**	**1.52254E+12**	**52248**
create new keys Matrix	**1.52254E+12**	**1.52254E+12**	**261673**
do-ranking	**1.52254E+12**	**1.52254E+12**	**626497**
final-ranking	**1.52254E+12**	**1.52254E+12**	**66009**
Top100	**1.52254E+12**	**1.52254E+12**	**26902**
Total Time			3470922

PERFORMANCE ANALYSIS WITH ADJECENCY LIST REPRESENTATION

ANALYSIS	Adjacency list(From Assignment 3 logs)	Row-Column	Column-Row
6 machines	3894	**4176**	**5678**
11 machines	2260	**2440**	**3320**
It's quite surprising that the matrix based approach has taken longer. This could be because the partitioning has taken more time to happen in this case as the data is being divided into many machines.			