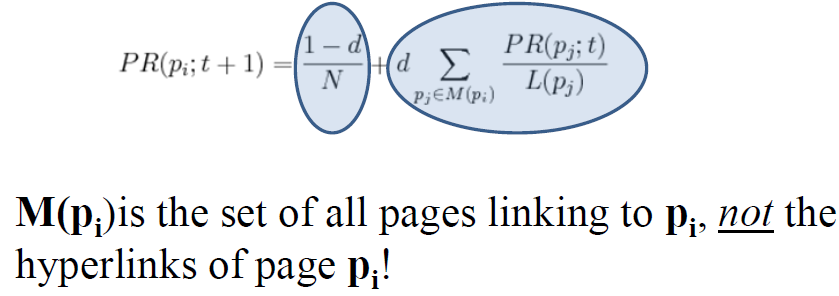
**Page Rank Algorithm**

For page rank algorithm, we start with a number of web pages, where each page points to another web page. Each web page pointing towards a another web page is considered a vote for the another web page. More the votes for a web page, the higher the rank for the web page. Google came with a formula for calculating the web page



We are going to compute the Page Rank for a web page using map reduce. We are going to being with **(URL, list-of-URLs)** where list-of-URLs is the outlinks from the webpage. We are going to transform the above expression to **(URL, (PageRank, list-of-URLs))** where **PageRank is the initial guess, usually 0.5 or 0.1**.

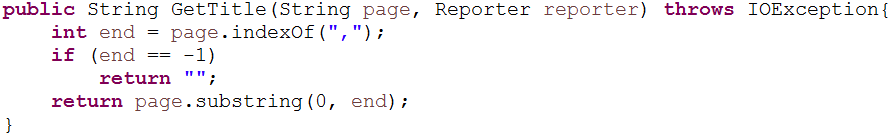
The above expression will act as input to the **Map**, the map for each ***u*** in the **list-of-URLs** will output **(u, URL, PR(URL)/Length of list-of-URLs)** i.e., the output link, with the voter, the page rank of the voter divided by number of output urls from the web page, the large the page rank of the URL, the more support to the web page will be give and we are going to output the **(URL, list-of-URLs)**.

Each Web page’s hyperlinks (outlinks, or votes) are “pivoted”, and we now tally up each Web page’s inlinks (voters) instead. In the **Reducer**, the input will be (URL, list-of-URLs) and many URLs pairs and it going to calculate the **(URL, (new PR, list-of-URLs))**. We are going to aggregate the votes from each voter and then going to multiply the aggregated value with the dampening factor. We are also going to add the the probability of the direc typing the URL and reaching the web page. The sum of the above expression will be the page rank of the web page. We are going to iterate through this mapper and reducer part couple of times and will observe that page rank is converging, after a while page rank will be constant and that will be the final page rank of the web page.

**Code Explanation**

For implementing the PageRank Algorithm, we have the concept it three parts which are named as **Graph Builder, Iterator and Viewer**.

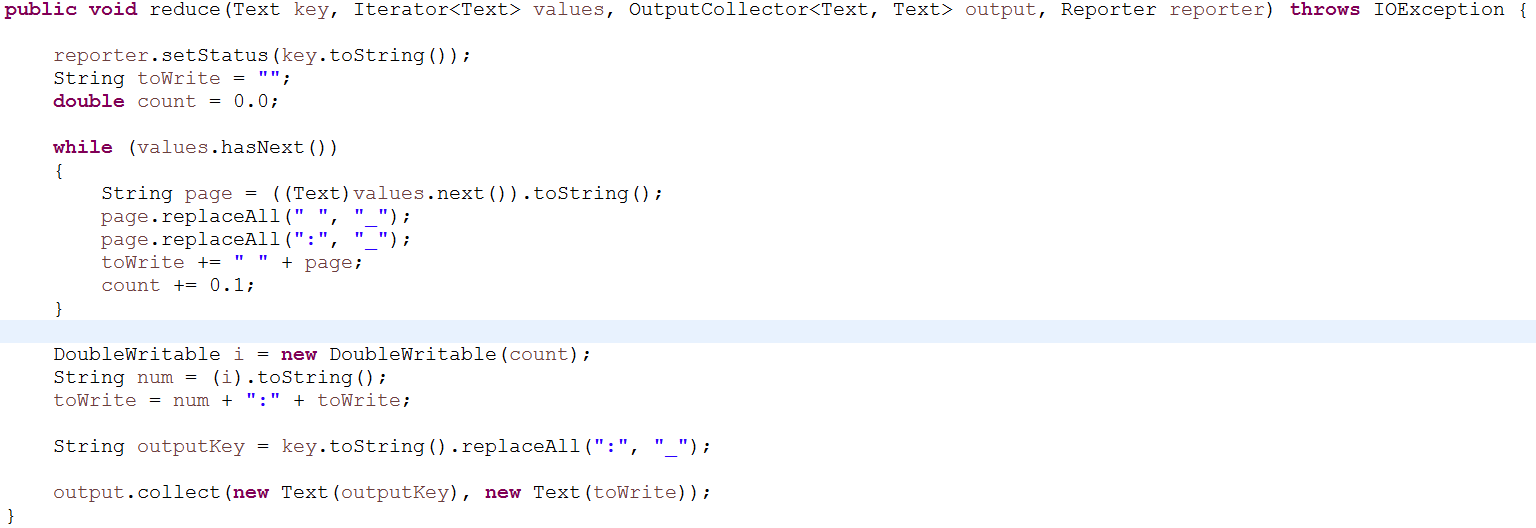
In **GraphBuilder Mapper**, we are going to iterate through each line of the shredded text file and **extract the URL** of the web page and **related outlinks of that URL**. In each line after every URL there is comma (,) and outlinks are written within **[[ ]]**. Below image is the function to get the title:



Below is the image to get the outlinks in the line. If in a particular outlink there is space, we are replacing it with **“\_”**. Since we are using space and colon later for manipulation, it is good practice to remove them now, otherwise later they will be considered as different url instead of one. We are going to output the tile as key and outlinks separated by space as value of the mapper.

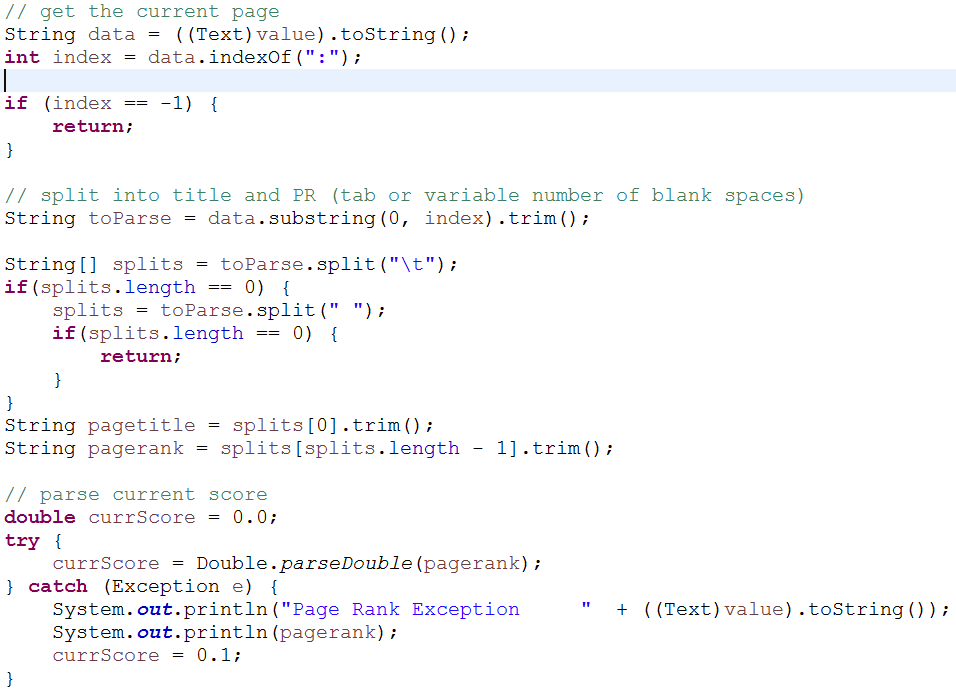


In the **GraphBuilder Reducer**, we are removing the space and colon from the outlinks and giving each URL a page rank as 0.1 (it is guessed one). The output of reducer will be **(URL, assumed page rank of the URL and the space separated outlinks)** and format will be **[URL PR: list-of-URLs]**.

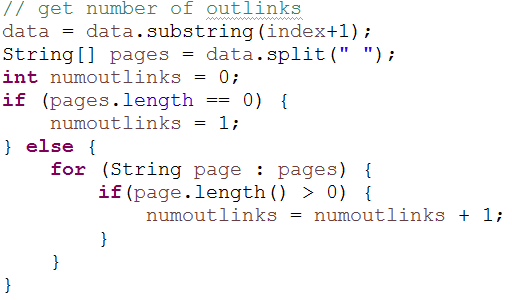


The output of the Graph Builder MapReduce will be considered as input for the species iterator MapReduce. The SpeciesIterator will receive the input as **URL, Page Rank and list-of-URLs** from the Graph Builder.

In the **SpeciesIterator Mapper,** we are going to split the input based on colon **“:”**, where the left part is the URL and page rank and the right part will be the list of the URL’s from that web page. If there is not **“:”**in the line, we are not going to considered that line. Now we are going to split the left part of the above part based on space or tab **Page Title and Page Rank.**

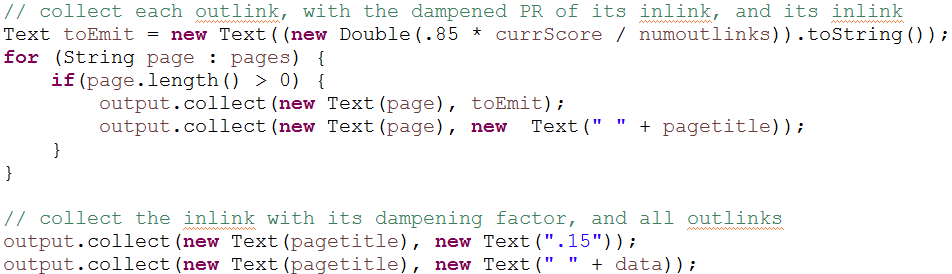


Now we are left with the string of the outlinks separated by **“ “,**  we are going to compute the number of the outlinks from the web page.

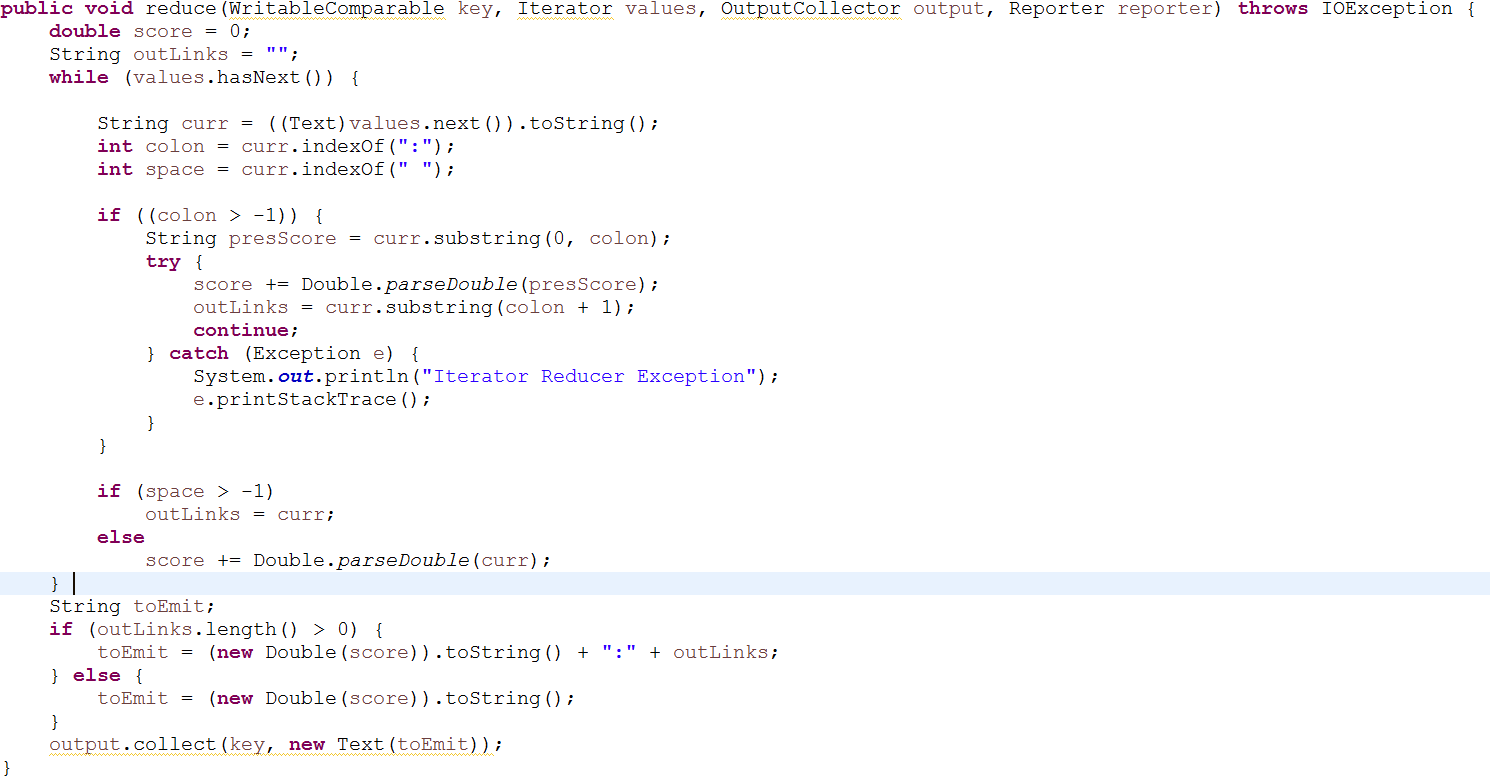


We are going to compute the **Silver surfer formula**, using the above values. We are going to assume the dampening factor and multiply it with the (Page Rank of the page / Length of the list-of-URLs) i.e., **[d \* (PR \* num\_of\_outlinks)]**.

The output of the mapper will be the URL and computed page rank, URL and list-of-URLs (outlinks). We are also going to output the webpage page rank [(1 – d) / N], i.e., the probability of direct reaching the web page. These are the URLs which were input to the mapper and also output the list-of-URLs for these web pages.

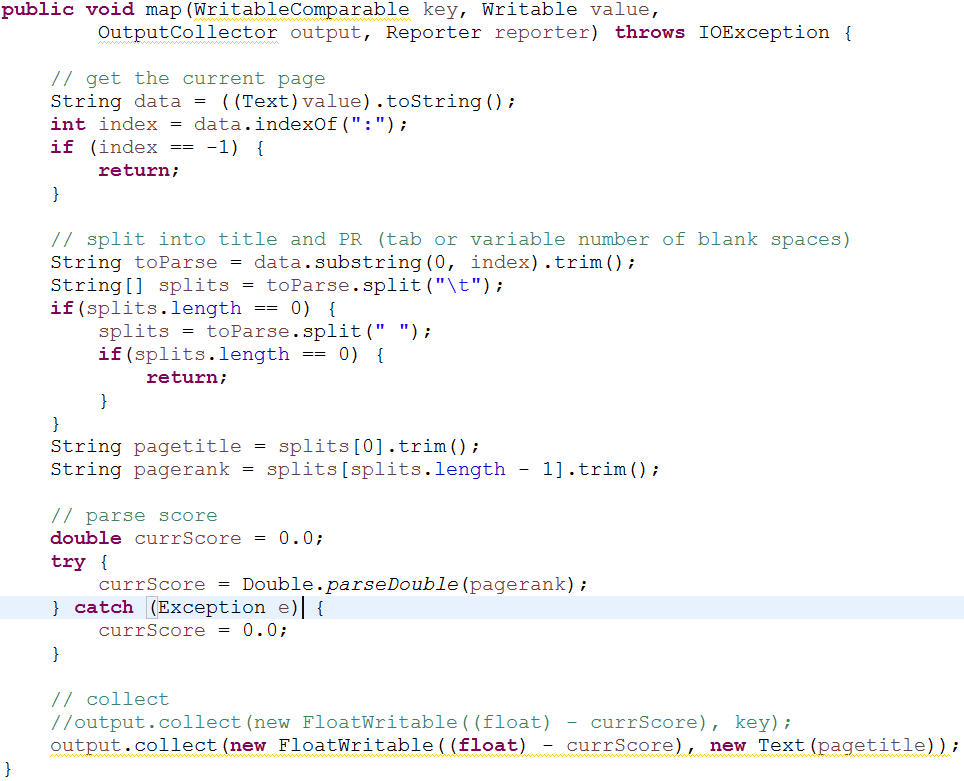


In the **SpeciesIterator Reducer**, we are splitting the data based on **“:”** and **“ “,**  to aggregate the score of the web page and separating the outlinks



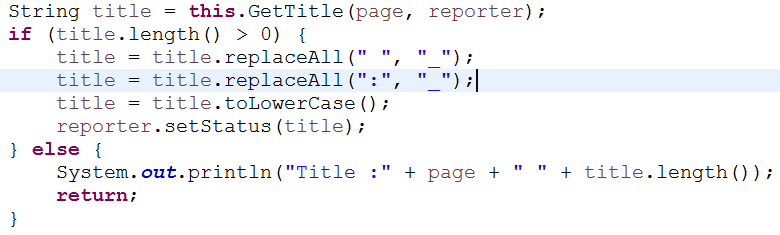
The output of the reducer will be the URL and the page rank of the URL. For species iterator, we are having 5 reducer tasks. This Species Iterator will be executed couple of times, till the time page rank converges completely. The converged page rank is going to act as input for Species Viewer.

In the **SpeciesViewer Mapper**, we are going to take the input from the Species Iterator, we are going to make the page rank as key and page title i.e., URL as value. We are going to make page rank as negative, so that when mapper sort the value, we can have the highest rank at the top. The output of the species viewer we conclude the page rank algorithm.

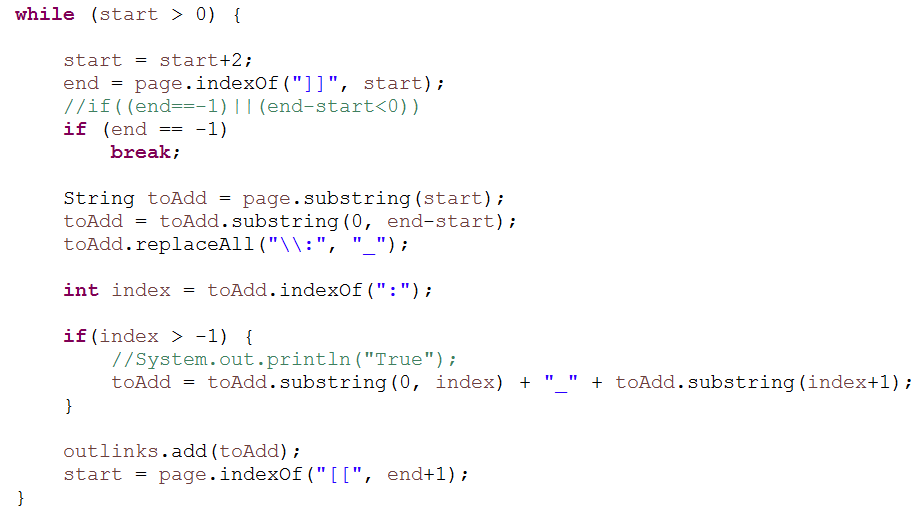


**Cleaning Performed in the Code**

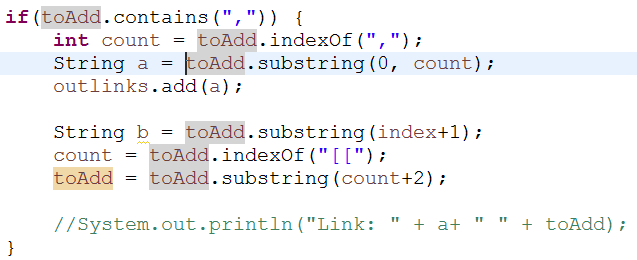
* Removed the spaces and colon from the title, as earlier it was breaking URL like **Homo sapiens** into two and giving a page rank for both the URL considering them as different and it was also changing the page rank of other web pages as well. **I replace the “ “ and “:” with “\_”.**



* Removed the spaces and colon from the list-of-URLs as, at many places we are splitting the data based on “ “ and “:”. So having the presence of these punctuation in the text result into the wrong computation and we were getting output as Key as Integer and value as Integer. There were presence of some different locale in the text which has “:” between the given URL, it was splitting them and they both were considered as outlink which is not the actual case, after replacing them those discrepancies were removed.



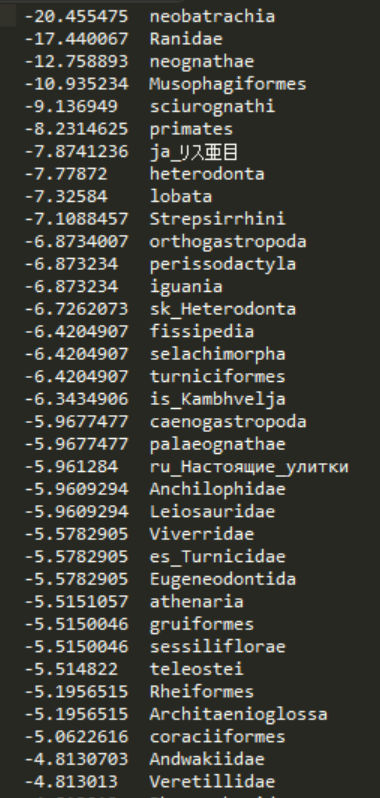
* There was few lines where there are open brackets but that are not closing, we have to take care of those cases as well. What eventually is happening is, the present code is considering one long string which has 2 outlinks as one. The code below is taking care of that thing, instead of closing square bracket there is a “,”. I am using that to retrieve the URL



**Result**

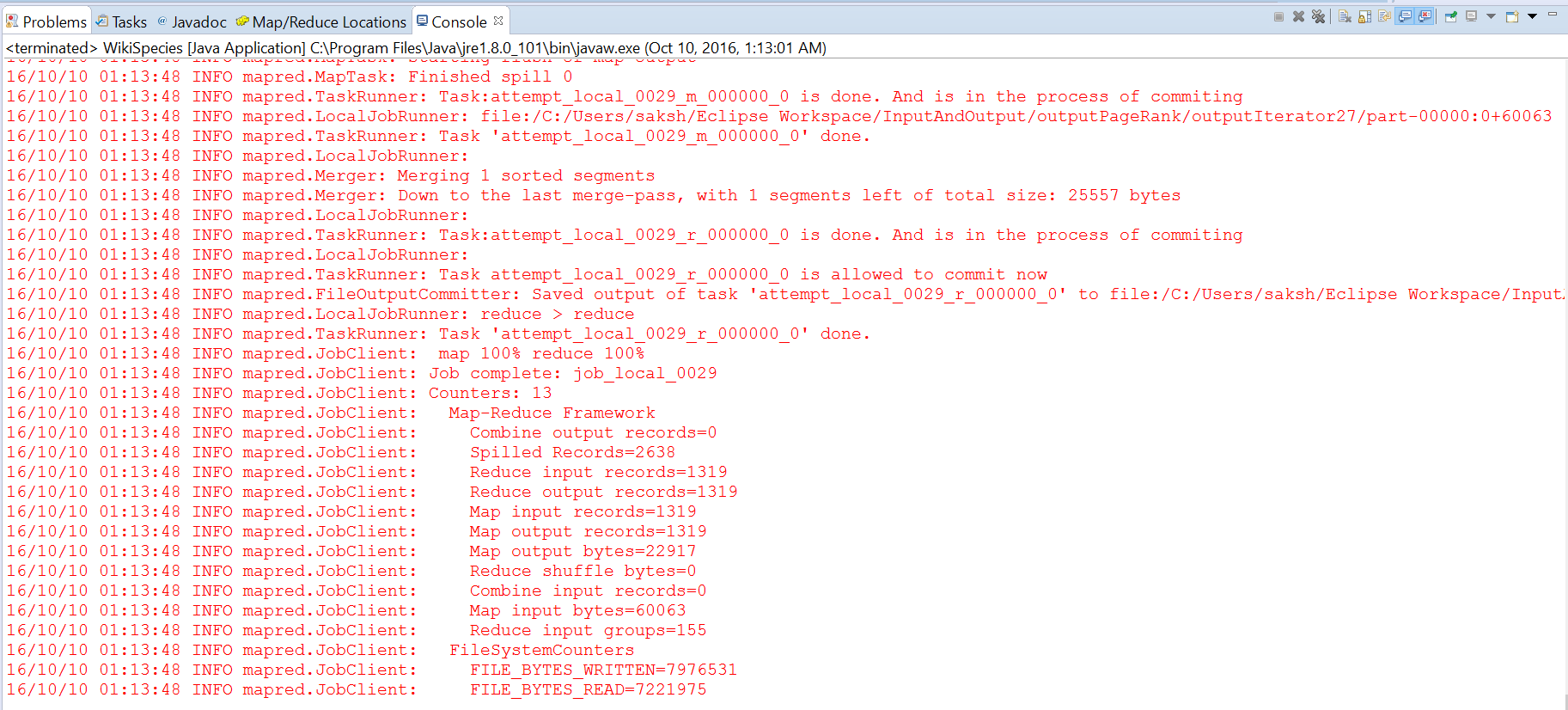
The data is converging; this has been observed in the result. I have run Species Iterator for 27 times and observed the value of URL “**reptilia**”. For 10th iteration the value is 2.79, for 11th iteration the value is 3.0, for 18th iteration the value is 3.49, for 22nd iteration the value is 3.61 and for 26th Iteration the value is 3.68 and the value in the final viewer is 3.70. This implies that page rank is converging for URLs. The assumed initial rank is 0.1 and dampening factor is 0.85. The maximum page rank which we are getting is 20.45, although these ranks should be less than 1 as these are the probabilities but for that we need to know and consider the probabilities appropriately.

**Final Output for Standalone Mode**



**Snapshots**

**Standalone Execution – Console**



**Pseudo Mode Execution Snapshots**

Commands to create java class and jar file and execute the jar on the Hadoop pseudo mode

We are sitting in a directory two cd inside where hadoop-0.20 folder is present.

1. Making the directory to store the class files

***mkdir -p wikispecies\_classes***

1. Compiling the java class to make class file of the java class

***javac -classpath ../../hadoop-0.20.2/hadoop-0.20.2-core.jar -d wikispecies\_classes \*.java***

1. Making a jar file with name wikispecies.jar

***jar cvf wikispecies.jar -C wikispecies\_classes/ .***

1. Execution of the wikispecies.jar in the pseudo distributed mode. It take 4 arguments, the first as number of iterations, second as the input folder from where it is going to read the input txt file, third as intermediary folder to store the output of the iterations and fourth as the final output folder to store the result of Species Viewer

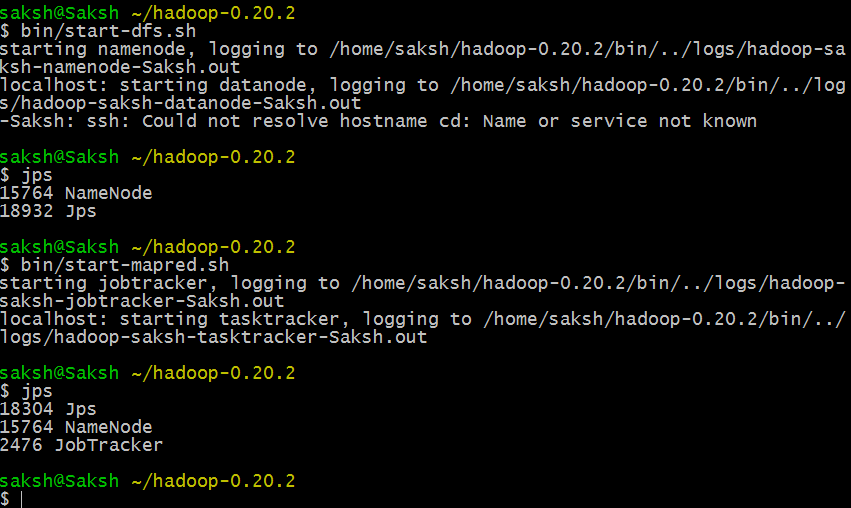
***../../hadoop-0.20.2/bin/hadoop jar wikispecies.jar com.species.Driver.WikiSpecies 27 inputPageRank outputItr output***

1. Putting the txt file on the hdfs. Because execution in the pseudo distributed mode, look for the file on the hdfs://localhost:9000/

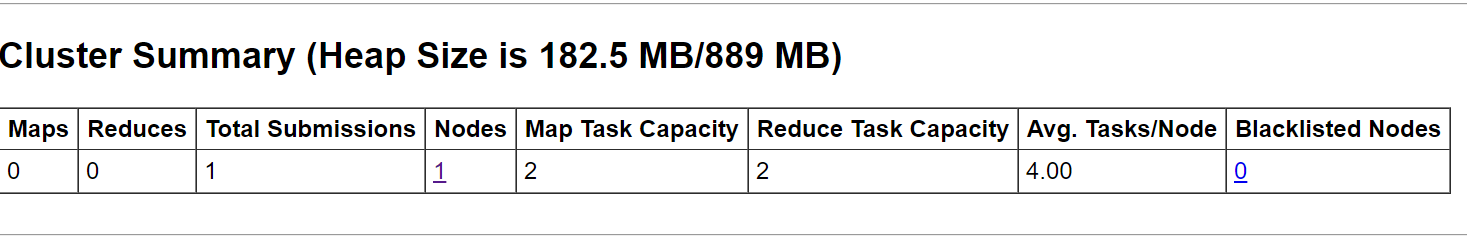
***../../hadoop-0.20.2/bin/Hadoop fs -put "C:/cygwin64/home/saksh/HadoopWorkspace/PageRank/inputPageRank/species\_medium.txt" inputPageRank***

1. To get the file from the hdfs server to local machine is

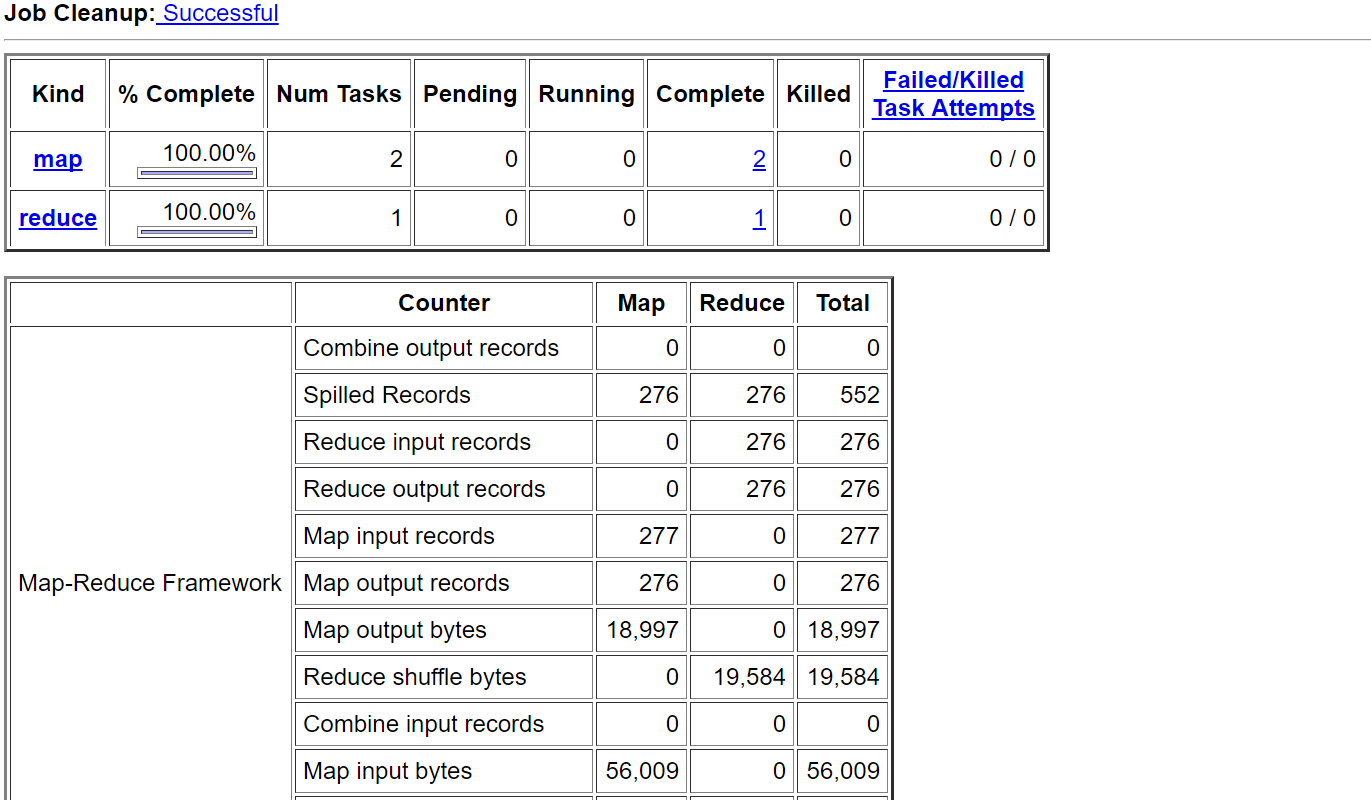
***../../hadoop-0.20.2/bin/hadoop fs -copyToLocal outputItr\* "C:\cygwin64\home\saksh\Hadoop Workspace\PageRank\outputFile"***

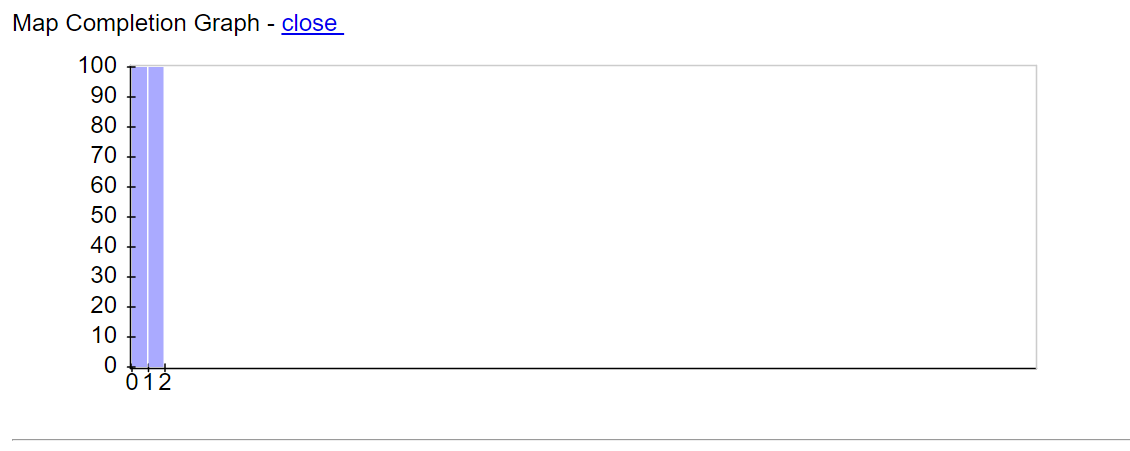


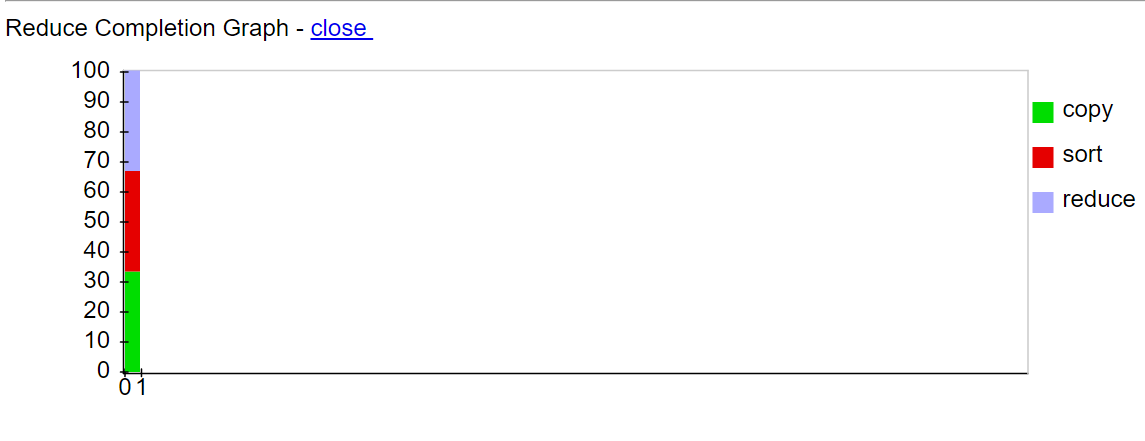
**Localhost:50030**



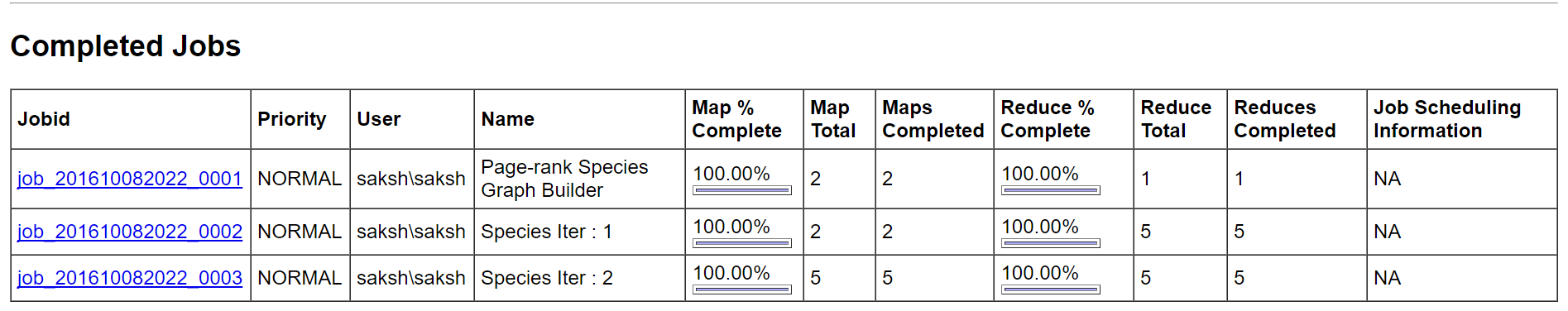
**Job Status**



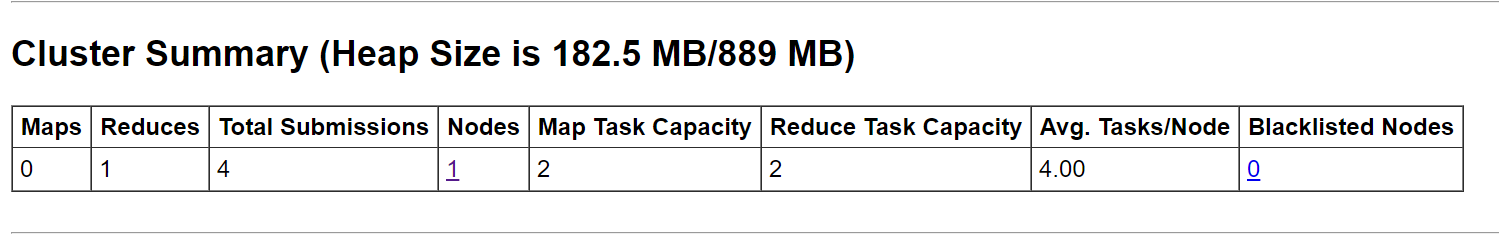


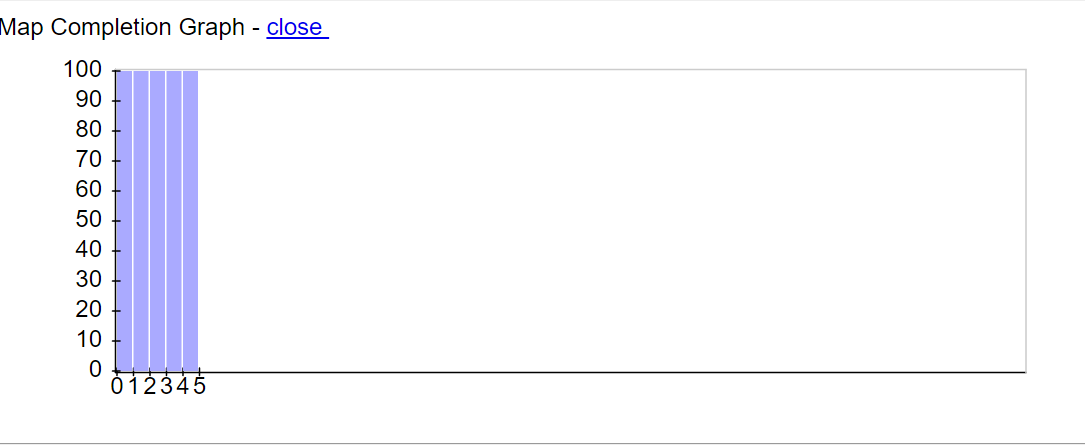


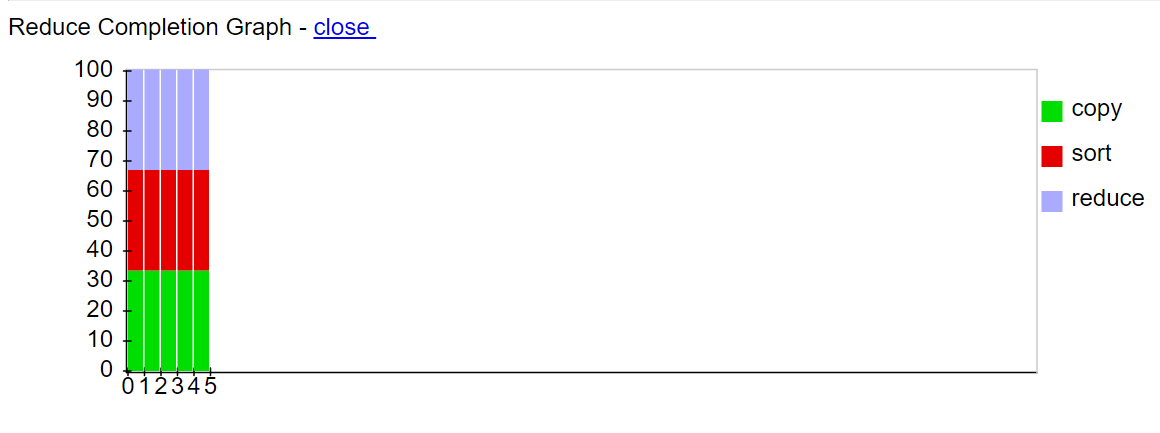
**Completed Job ID’s**



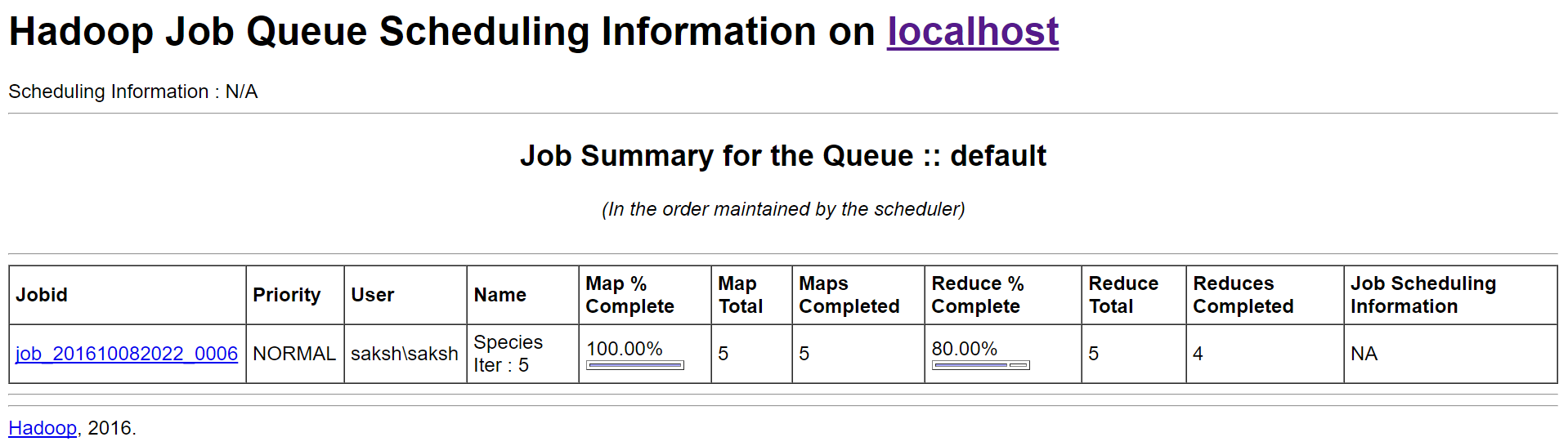
**Summary of localhost in middle of execution**



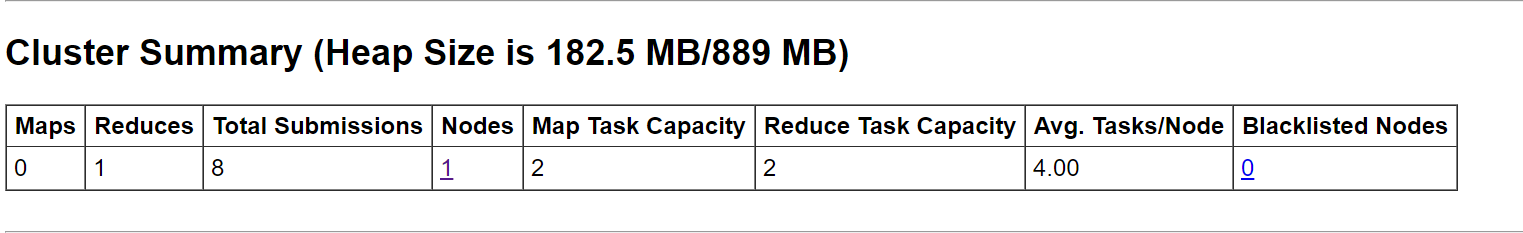




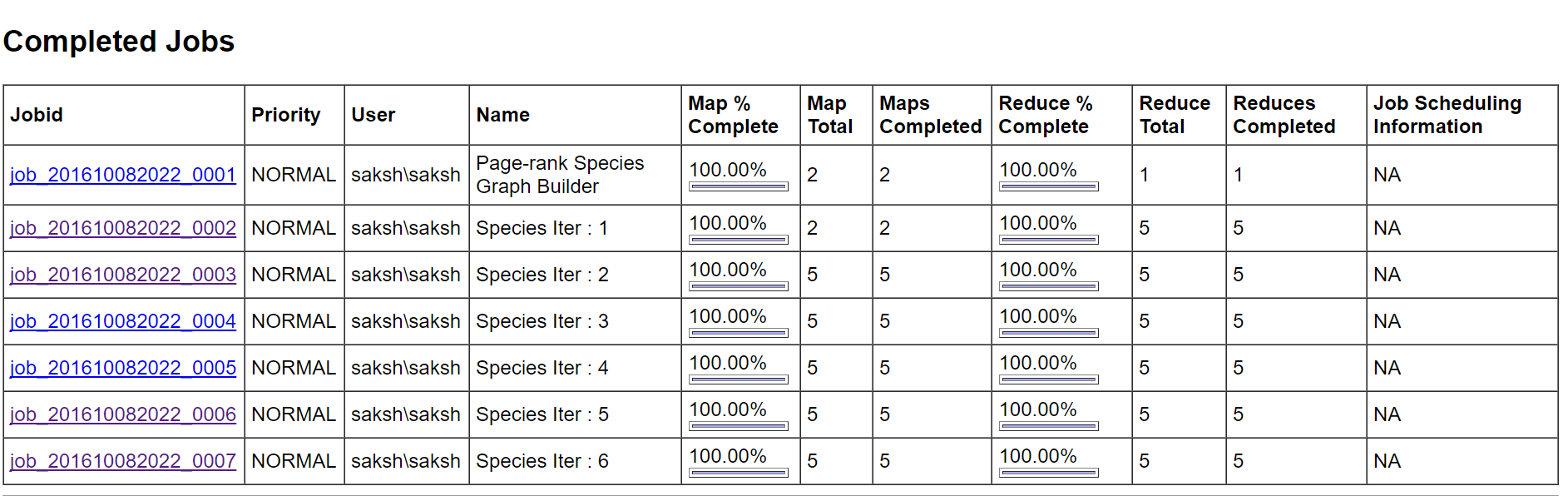
**Job Queued is Iteration 5 of the Species Iteration**



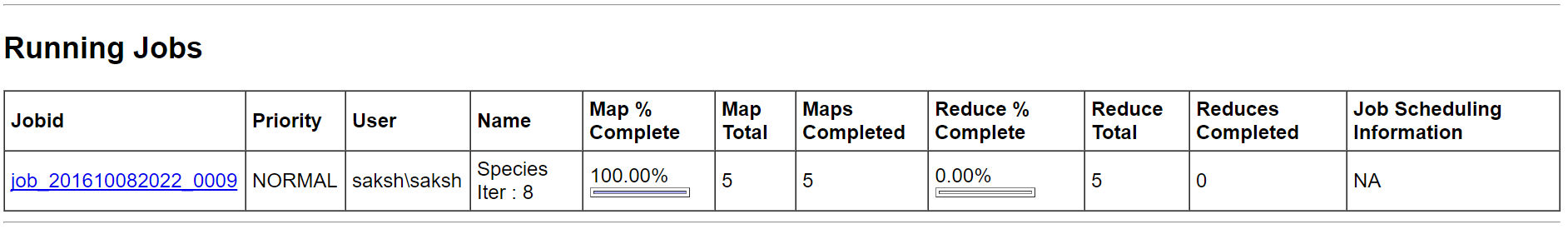
Cluster information of localhost:50030

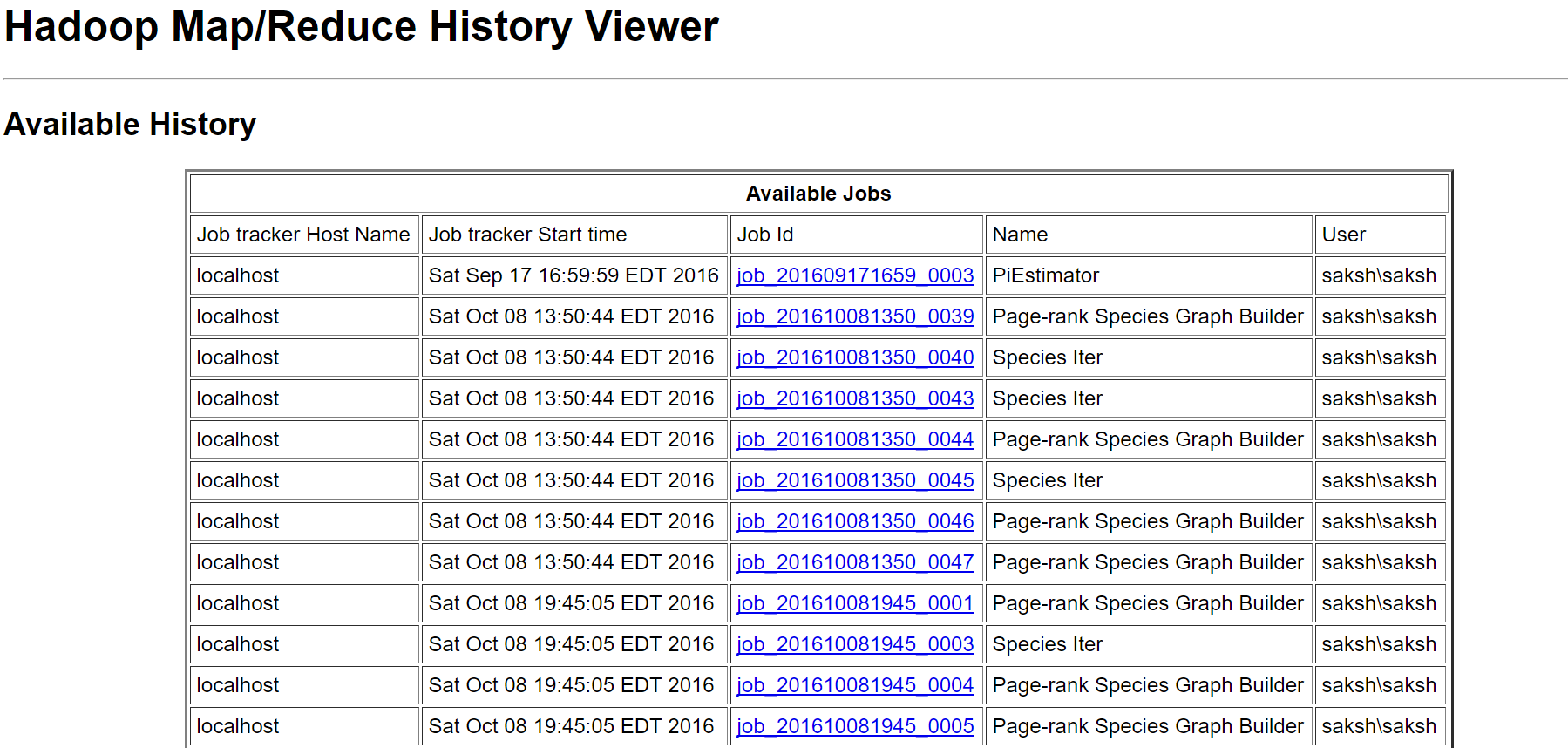


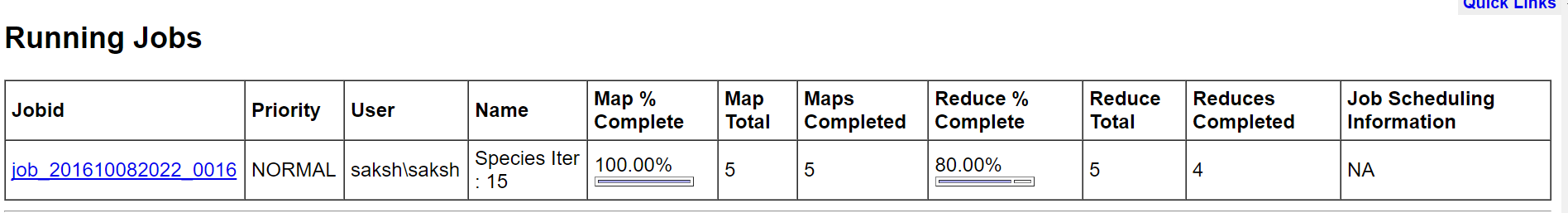
Completed Job Id’s



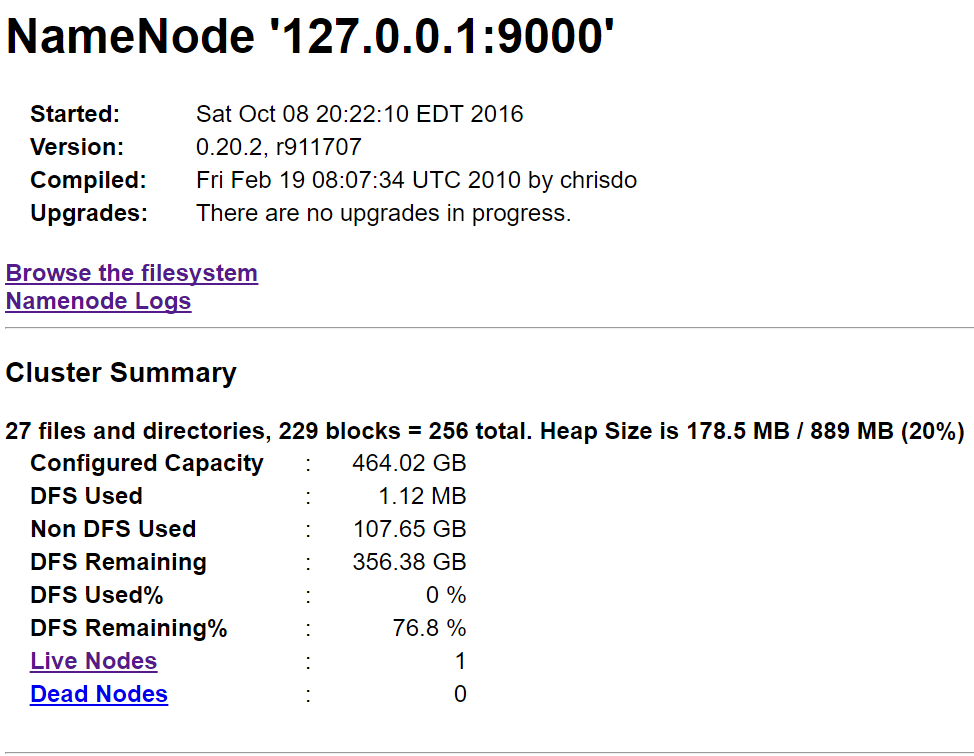
**Running Job is Iteration 8**

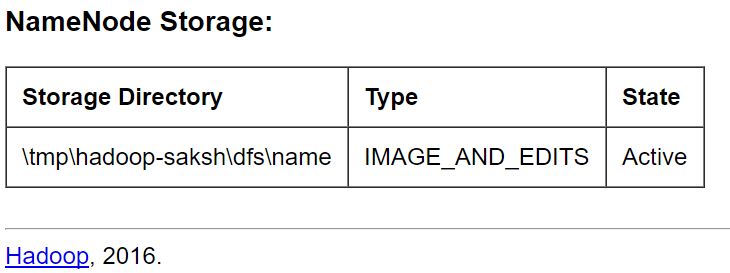


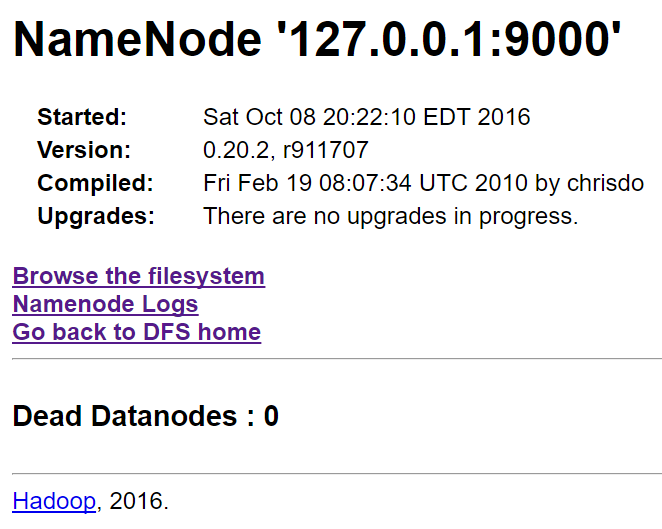


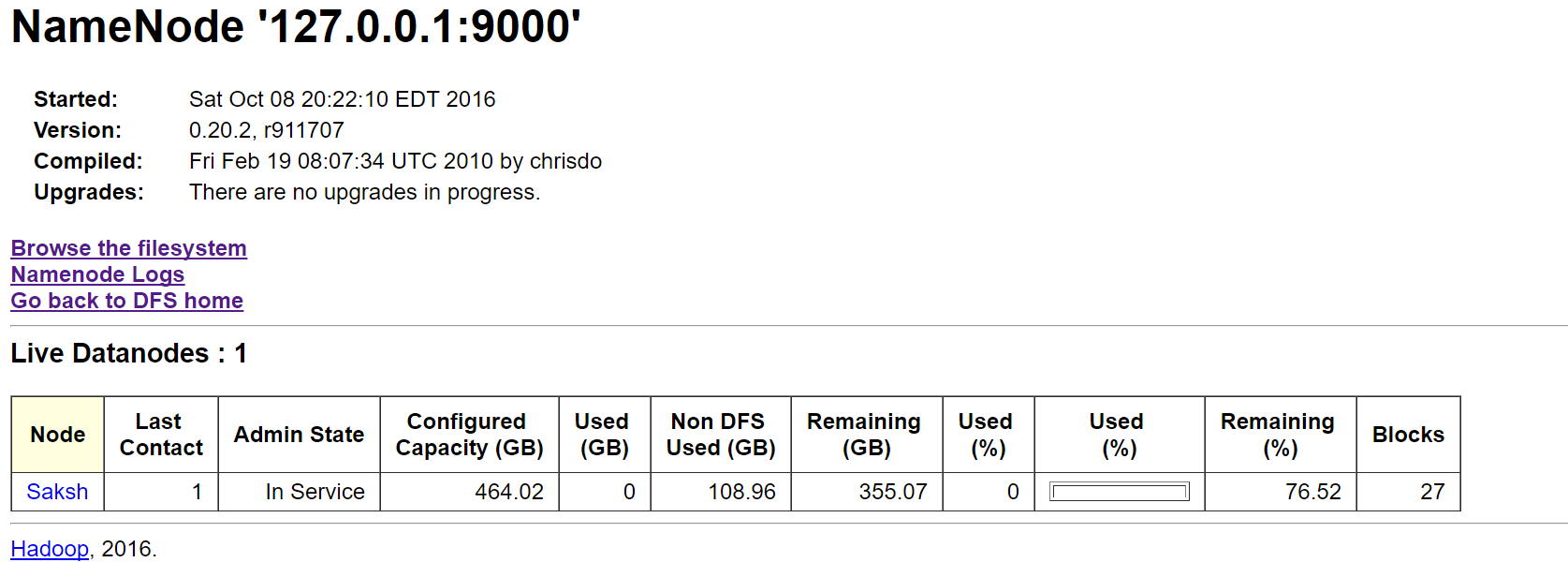


**Localhost:50070**

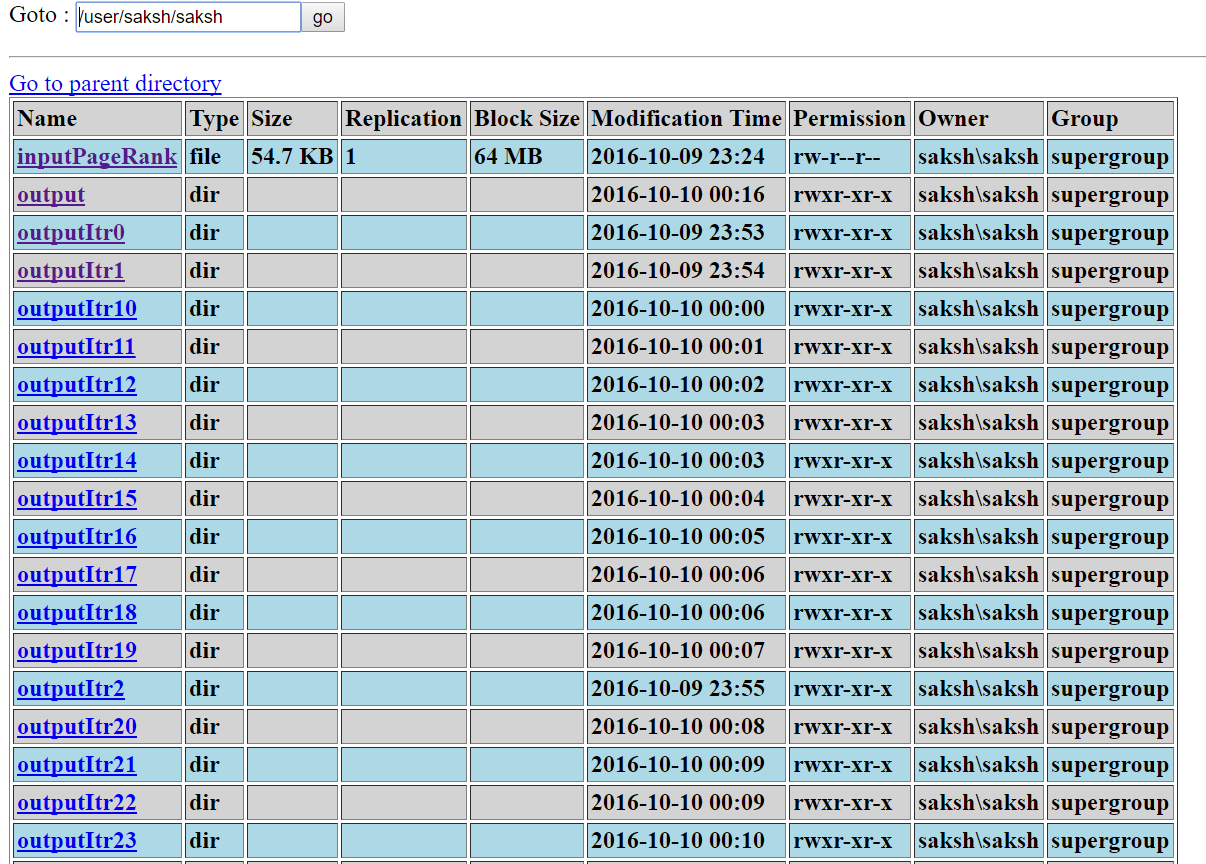




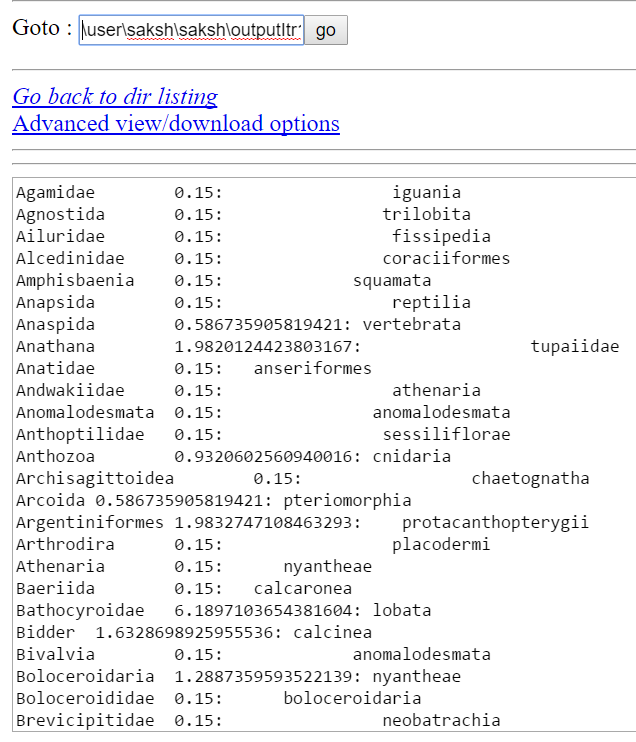




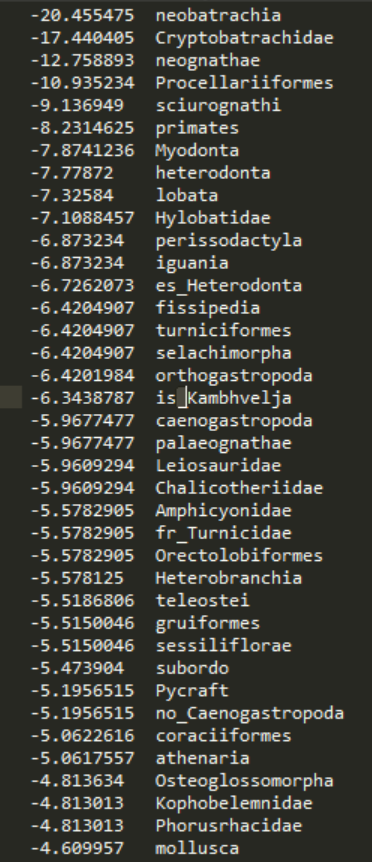
**Output Directory Structure**



**Output of the 17th Iteration**



**Final output of Pseudo Mode**



**For other output you can refer the output folder**