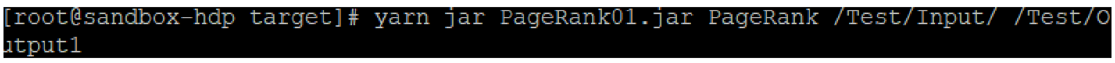
Report 107062546 楊仲愷

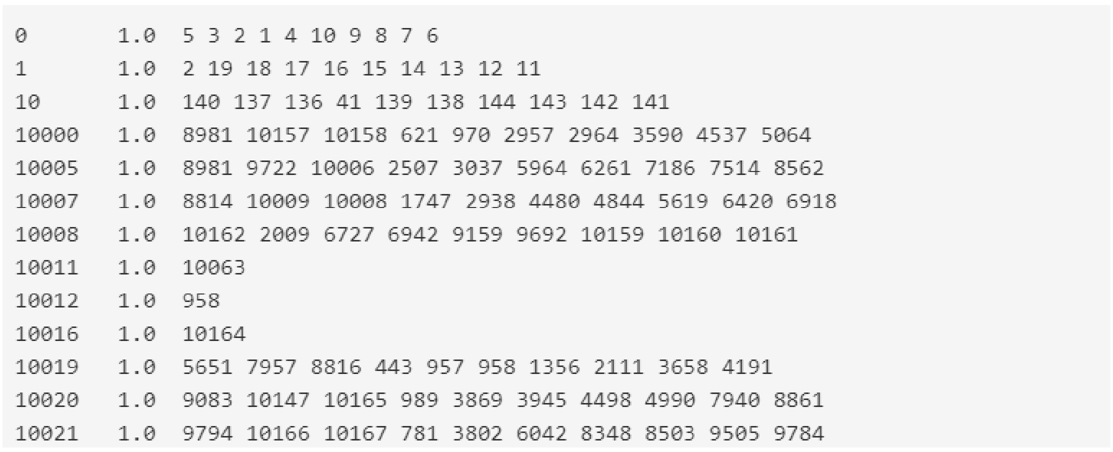
Java File, Class Name: PageRank01, PageRank
PageRank02, PageRank
PageRank03, PageRank

First, I deal with the data into websites, rank, and links. Therefore, type this command and the input path is Input, output path is Output1.

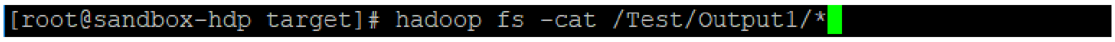


```
[root@sandbox-hdp target]# yarn jar PageRank01.jar PageRank /Test/Input/ /Test/O
utput1
```

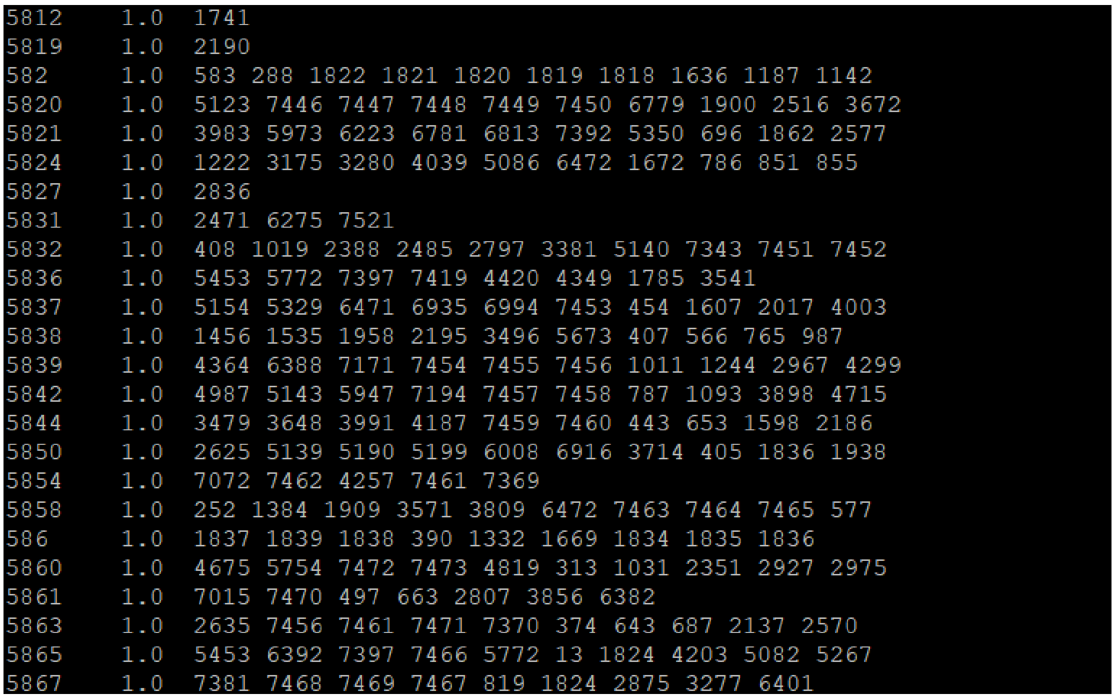
Then, we will see the data like this.



```
0       1.0  5 3 2 1 4 10 9 8 7 6
1       1.0  2 19 18 17 16 15 14 13 12 11
10      1.0  140 137 136 41 139 138 144 143 142 141
10000   1.0  8981 10157 10158 621 970 2957 2964 3590 4537 5064
10005   1.0  8981 9722 10006 2507 3037 5964 6261 7186 7514 8562
10007   1.0  8814 10009 10008 1747 2938 4480 4844 5619 6420 6918
10008   1.0  10162 2009 6727 6942 9159 9692 10159 10160 10161
10011   1.0  10063
10012   1.0  958
10016   1.0  10164
10019   1.0  5651 7957 8816 443 957 958 1356 2111 3658 4191
10020   1.0  9083 10147 10165 989 3869 3945 4498 4990 7940 8861
10021   1.0  9794 10166 10167 781 3802 6042 8348 8503 9505 9784
```



```
[root@sandbox-hdp target]# hadoop fs -cat /Test/Output1/*
```



```
5812    1.0  1741
5819    1.0  2190
582     1.0  583 288 1822 1821 1820 1819 1818 1636 1187 1142
5820    1.0  5123 7446 7447 7448 7449 7450 6779 1900 2516 3672
5821    1.0  3983 5973 6223 6781 6813 7392 5350 696 1862 2577
5824    1.0  1222 3175 3280 4039 5086 6472 1672 786 851 855
5827    1.0  2836
5831    1.0  2471 6275 7521
5832    1.0  408 1019 2388 2485 2797 3381 5140 7343 7451 7452
5836    1.0  5453 5772 7397 7419 4420 4349 1785 3541
5837    1.0  5154 5329 6471 6935 6994 7453 454 1607 2017 4003
5838    1.0  1456 1535 1958 2195 3496 5673 407 566 765 987
5839    1.0  4364 6388 7171 7454 7455 7456 1011 1244 2967 4299
5842    1.0  4987 5143 5947 7194 7457 7458 787 1093 3898 4715
5844    1.0  3479 3648 3991 4187 7459 7460 443 653 1598 2186
5850    1.0  2625 5139 5190 5199 6008 6916 3714 405 1836 1938
5854    1.0  7072 7462 4257 7461 7369
5858    1.0  252 1384 1909 3571 3809 6472 7463 7464 7465 577
586     1.0  1837 1839 1838 390 1332 1669 1834 1835 1836
5860    1.0  4675 5754 7472 7473 4819 313 1031 2351 2927 2975
5861    1.0  7015 7470 497 663 2807 3856 6382
5863    1.0  2635 7456 7461 7471 7370 374 643 687 2137 2570
5865    1.0  5453 6392 7397 7466 5772 13 1824 4203 5082 5267
5867    1.0  7381 7468 7469 7467 819 1824 2875 3277 6401
```

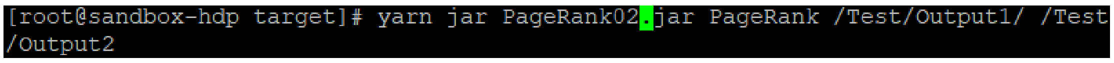
Here is the map. I process the original string to a website and its out links. Therefore, I set the website to key, and out links to value.

```
 public void map(Object key, Text value, Context context
 ) throws IOException, InterruptedException {

     StringTokenizer st = new StringTokenizer(value.toString());
     while (st.hasMoreTokens()) {
         word1.set(st.nextToken());
         word2.set(st.nextToken());

         word3 = word1;
         word4 = word2;

         context.write(word3, word4);
     }
 }
```

Consequently, I reduce the website and its out links, and add the rank into the line. Then, we can get the line like “website   rank   out links“.

```
  public void reduce(Text key,Iterable<Text> values,Context context) throws
      String str = new String();

      str = "1.0 " + str;

      for (Text val : values) {
          str = str + " " +val.toString();
      }
      value.set(str);
      key.set(key);
      context.write(key, value);
  }
```

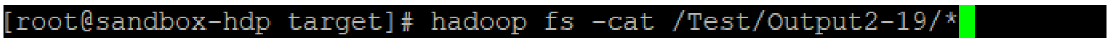
Secondly, let’s process the data with the function for twenty times.



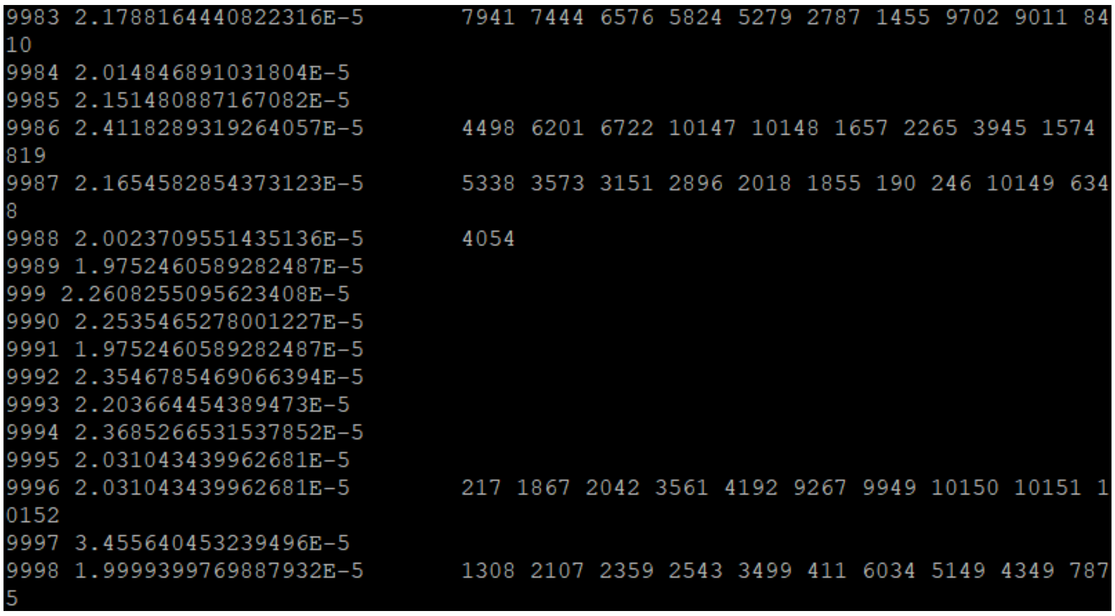
```
[root@sandbox-hdp target]# yarn jar PageRank02.jar PageRank /Test/Output1/ /Test
/Output2
```

And we will see there are twenty directories from Output2-0 to Output2-19. They represent which times they process. We open Output2-19 which means the twenty times processed data and will see results below.

```
0 3.827506823847662E-5   5 3 2 1 4 10 9 8 7 6
1 2.72964916684691E-5    2 19 18 17 16 15 14 13 12 11
10 2.9995348573988325E-5        140 137 136 41 139 138 144 143 142 141
100 2.8516524322774366E-5
1000 2.464396856814206E-5
10000 2.0034758060957863E-5     8981 10157 10158 621 970 2957 2964 3590 4537 5064
10001 2.0034758060957863E-5
10002 1.9998414424906583E-5
10003 1.9998414424906583E-5
10004 2.1633750274031956E-5
10005 1.838911364472232E-5      8981 9722 10006 2507 3037 5964 6261 7186 7514 8562
10006 2.1380655072016456E-5
10007 1.838911364472232E-5      8814 10009 10008 1747 2938 4480 4844 5619 6420 6918
10008 1.9860242736300108E-5     10162 2009 6727 6942 9159 9692 10159 10160 10161
```



```
[root@sandbox-hdp target]# hadoop fs -cat /Test/Output2-19/*
```



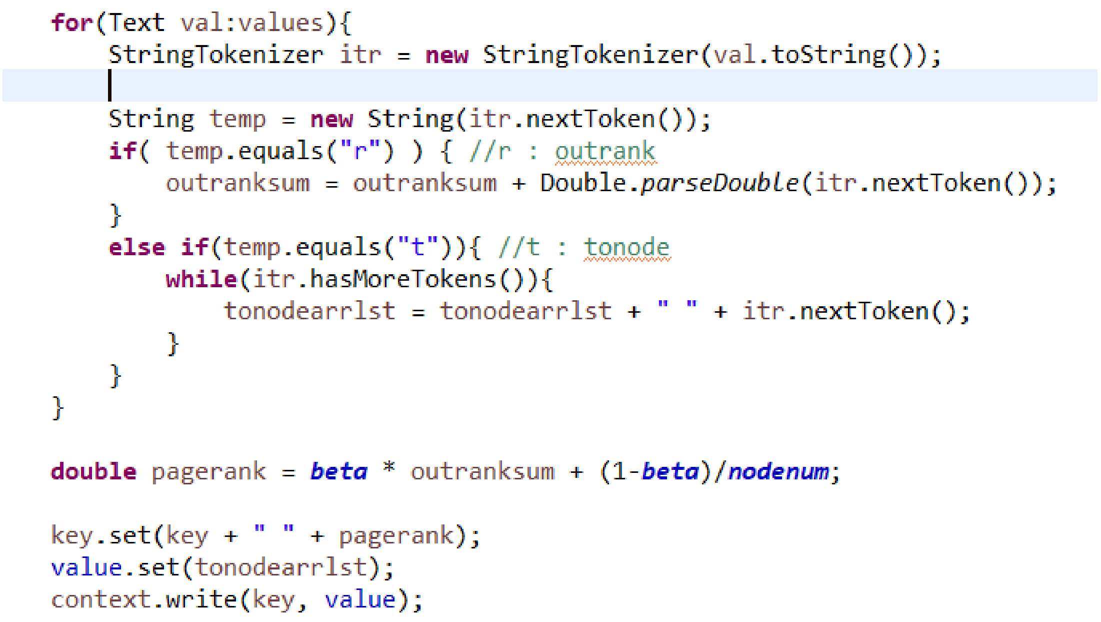
```
9983 2.1788164440822316E-5      7941 7444 6576 5824 5279 2787 1455 9702 9011 84
10
9984 2.014846891031804E-5
9985 2.151480887167082E-5
9986 2.4118289319264057E-5      4498 6201 6722 10147 10148 1657 2265 3945 1574
819
9987 2.1654582854373123E-5      5338 3573 3151 2896 2018 1855 190 246 10149 634
8
9988 2.0023709551435136E-5      4054
9989 1.9752460589282487E-5
999 2.260825509562340 8E-5
9990 2.2535465278001227E-5
9991 1.9752460589282487E-5
9992 2.354678546906 6394E-5
9993 2.203664454389473E-5
9994 2.3685266531537852E-5
9995 2.031043439962681E-5
9996 2.031043439962681E-5      217 1867 2042 3561 4192 9267 9949 10150 10151 1
0152
9997 3.455640453239496E-5
9998 1.9999399769887932E-5      1308 2107 2359 2543 3499 411 6034 5149 4349 787
5
```

I set out links and their rank to keys and values, and then set from links and out links to another keys and values. Specially, I add ‘t’ and ‘r’ to represent to and out.

```
    if(itr.hasMoreTokens()) {
        fromnode.set(itr.nextToken());
        rank = Double.parseDouble(itr.nextToken());
    }


    while(itr.hasMoreTokens()){
        tonodelist.add(itr.nextToken());
    }

    outlink = rank / tonodelist.size();

    for(int i = 0; i<tonodelist.size(); i++){
        Text keys = new Text(tonodelist.get(i));
        Text value = new Text("r " + outlink);

        context.write(keys, value);
    }

    for(int i = 0; i<tonodelist.size(); i++){
        tonode = tonode + " " + tonodelist.get(i);
    }

    context.write(fromnode, new Text(tonode));
```

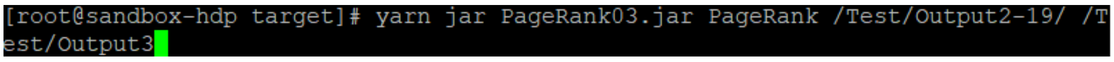
Here, we can see I take advantage of the string to determine whether it is out or to.

Then, let’s put data into the formulation $r_j = \sum_{i \to j} \beta \frac{r_i}{d_i} + (1-\beta)\frac{1}{N}$. And we will get the processed rank. Data are like “website rank   out links”.



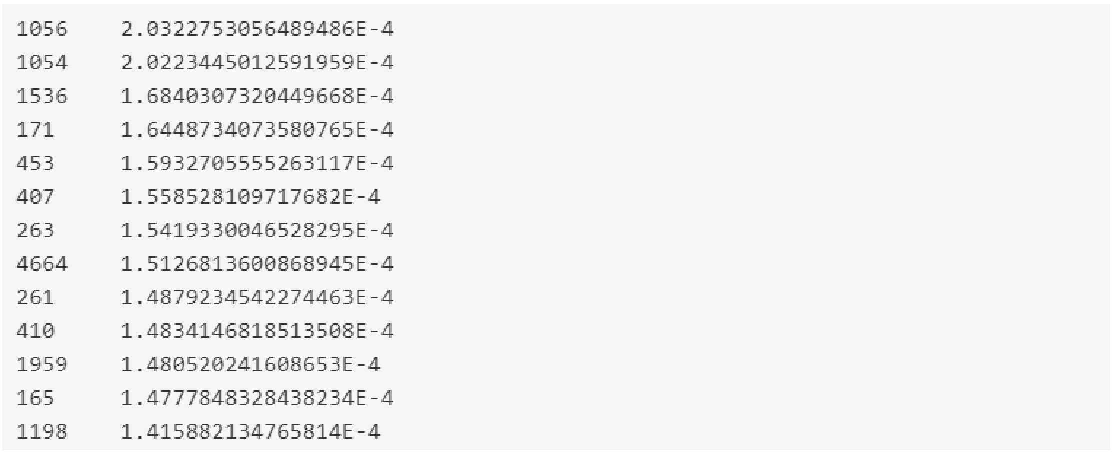
```
for(Text val:values){
    StringTokenizer itr = new StringTokenizer(val.toString());

    String temp = new String(itr.nextToken());
    if( temp.equals("r") ) { //r : outrank
        outranksum = outranksum + Double.parseDouble(itr.nextToken());
    }
    else if(temp.equals("t")){ //t : tonode
        while(itr.hasMoreTokens()){
            tonodearrlst = tonodearrlst + " " + itr.nextToken();
        }
    }
}

double pagerank = beta * outranksum + (1-beta)/nodenum;

key.set(key + " " + pagerank);
value.set(tonodearrlst);
context.write(key, value);
```

Finally, we got the processed data from PageRank02, and we can sort the results with descending because we can see the top ten websites from this.

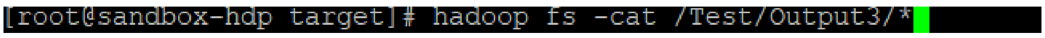


```
[root@sandbox-hdp target]# yarn jar PageRank03.jar PageRank /Test/Output2-19/ /Test/Output3
```

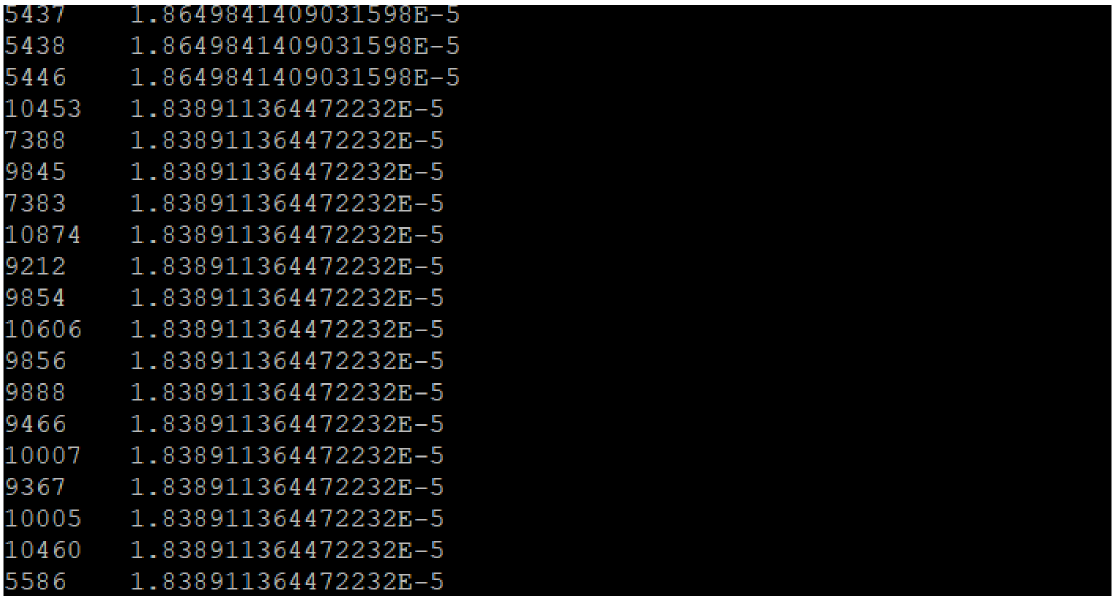
And we can the results like this which is sorted with descending.



```
1056    2.0322753056489486E-4
1054    2.0223445012591959E-4
1536    1.6840307320449668E-4
171     1.6448734073580765E-4
453     1.5932705555263117E-4
407     1.558528109717682E-4
263     1.5419330046528295E-4
4664    1.5126813600868945E-4
261     1.4879234542274463E-4
410     1.4834146818513508E-4
1959    1.480520241608653E-4
165     1.4777848328438234E-4
1198    1.415882134765814E-4
```



```
[root@sandbox-hdp target]# hadoop fs -cat /Test/Output3/*
```



```
5437    1.8649841409031598E-5
5438    1.8649841409031598E-5
5446    1.8649841409031598E-5
10453   1.838911364472232E-5
7388    1.838911364472232E-5
9845    1.838911364472232E-5
7383    1.838911364472232E-5
10874   1.838911364472232E-5
9212    1.838911364472232E-5
9854    1.838911364472232E-5
10606   1.838911364472232E-5
9856    1.838911364472232E-5
9888    1.838911364472232E-5
9466    1.838911364472232E-5
10007   1.838911364472232E-5
9367    1.838911364472232E-5
10005   1.838911364472232E-5
10460   1.838911364472232E-5
5586    1.838911364472232E-5
```

As a result, we can see the top ten websites from the results. They are separately 1056, 1054, 1536, 171, 453, 407, 263, 4664, 261, 410.

I set the rank and the website to separately key and value because when this reduce it will be sorted. However, output list will be ascending, but I want list which is descending. Thus, I multiple all the rank negative one, and the rank will be totally opposite.

```
Text node = new Text();
node.set(key.toString());

if(itr.hasMoreTokens())
{
    IntWritable intwritable = new IntWritable(Integer.parseInt(itr.nextToken().toString()));
    DoubleWritable doublewritable = new DoubleWritable(Double.parseDouble(itr.nextToken().toString())*-1) ;

    context.write(doublewritable, intwritable);

}
```

I multiple rank negative again and set website and link to key and value for correct output, and the rank will be not only original but also descending.

```
public void reduce(DoubleWritable key, Iterable<IntWritable> value, Context context)

    for(IntWritable val :value){
        key.set(key.get()*-1);

        context.write(val, key);
    }
}
```