

CentraleSupélec, 2017-2018

Programming languages and platforms

Group : Yuan ZHANG, Zhenyu ZOU



CentraleSupélec

Task Distribution

Question 2.7 :

- Code : Zhenyu ZOU
- Report : Zhenyu ZOU

Question 2.8 :

- Code : Yuan ZHANG
- Report : Zhenyu ZOU

Question 5.1 :

- Code : Yuan ZHANG
- Report : Yuan ZHANG, Zhenyu ZOU

Question 5.2 :

- Code : Yuan ZHANG
- Report : Yuan ZHANG, Zhenyu ZOU

Question 5.3 :

- Code : Zhenyu ZOU
- Report : Yuan ZHANG, Zhenyu ZOU

Question 2.7

ANNEE	PLANTATION, HAUTEUR
1935,	13.0
1854,	20.0
1862,	22.0
1906,	16.0
1784,	30.0
1860,	45.0
1840,	40.0
 ...	

Result of question 2.7

Question 2.8

USAF Code: 999999, Station's name: SAN ANGELO GOODFELLOW AFB ,	FIPS Country ID: US, Altitude: +0573.0
USAF Code: 999999, Station's name: BIGGS AFB ,	FIPS Country ID: US, Altitude: +1196.0
USAF Code: 999999, Station's name: AURORA BUCKLEY FIELD ANGB ,	FIPS Country ID: US, Altitude: +1725.8
USAF Code: 999999, Station's name: LUBBOCK WEST TEXAS AIR TERMIN ,	FIPS Country ID: US, Altitude: +0987.9
USAF Code: 999999, Station's name: ROSWELL MUNI AP ,	FIPS Country ID: US, Altitude: +1106.1
USAF Code: 999999, Station's name: EL PASO MUNICIPAL ARPT ,	FIPS Country ID: US, Altitude: +1193.6
USAF Code: 999999, Station's name: TUCUMCARI FAA AP ,	FIPS Country ID: US, Altitude: +1231.1
USAF Code: 999999, Station's name: ALBUQUERQUE MUNICIPAL ARPT ,	FIPS Country ID: US, Altitude: +1619.7
USAF Code: 999999, Station's name: CLAYTON MUNICIPAL AIRPARK ,	FIPS Country ID: US, Altitude: +1515.5
USAF Code: 999999, Station's name: RATON MUNICIPAL CREWS FIELD A ,	FIPS Country ID: US, Altitude: +1935.2
USAF Code: 999999, Station's name: PAMPA ,	FIPS Country ID: US, Altitude: +0985.1
USAF Code: 999999, Station's name: GUADALUPE PASS ,	FIPS Country ID: US, Altitude: +1663.0
USAF Code: 999999, Station's name: OTTO FAA AP ,	FIPS Country ID: US, Altitude: +1900.1
USAF Code: 999999, Station's name: ACOMITA FAA AP ,	FIPS Country ID: US, Altitude: +2007.1
USAF Code: 999999, Station's name: COLUMBUS ,	FIPS Country ID: US, Altitude: +1239.0
USAF Code: 999999, Station's name: SALT FLAT ,	FIPS Country ID: US, Altitude: +1132.9
USAF Code: 999999, Station's name: EL MORRO CAA AP ,	FIPS Country ID: US, Altitude: +2171.1
USAF Code: 999999, Station's name: ALAMOSA ARPT ,	FIPS Country ID: US, Altitude: +2298.5
USAF Code: 999999, Station's name: DENVER STAPLETON AIRFIELD ,	FIPS Country ID: US, Altitude: +1626.1
USAF Code: 999999, Station's name: EAGLE COUNTY AP ,	FIPS Country ID: US, Altitude: +1991.9
USAF Code: 999999, Station's name: GOODLAND RENNER FIELD ,	FIPS Country ID: US, Altitude: +1124.1
USAF Code: 999999, Station's name: GRAND JUNCTION WALKER FIELD ,	FIPS Country ID: US, Altitude: +1474.9

Result of question 2.8

Question 5.1 : TF-IDF

TF-IDF measures how important a word to a document or a group of documents. TF shows the frequency of a term in a document and IDF describes how much of information that this term provides.

This program includes :

1) java files

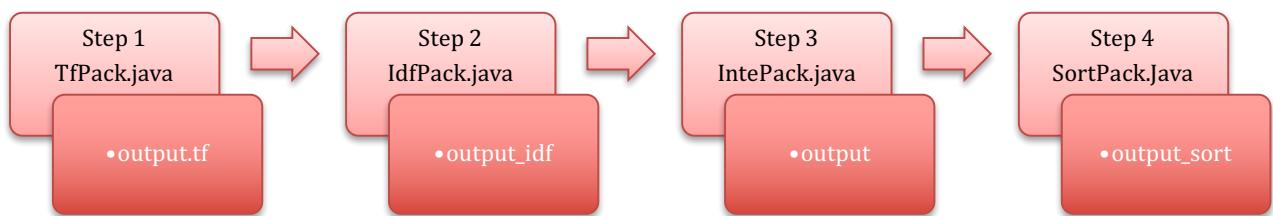
- TfIdf_Driver.java : main program
- TfPack.java : TF calculation
- IdfPack.java : IDF calculation
- IntePack.java : combine the result of TF&IDF and calculate TF-IDF
- SortPack.java : results sorting

2) 2 input files

- callwild
- defoe-robinson-103.txt

3) 4 output files

- output_tf
- output_idf
- output
- output_sort



• **Step 1 (TfPack.java)**

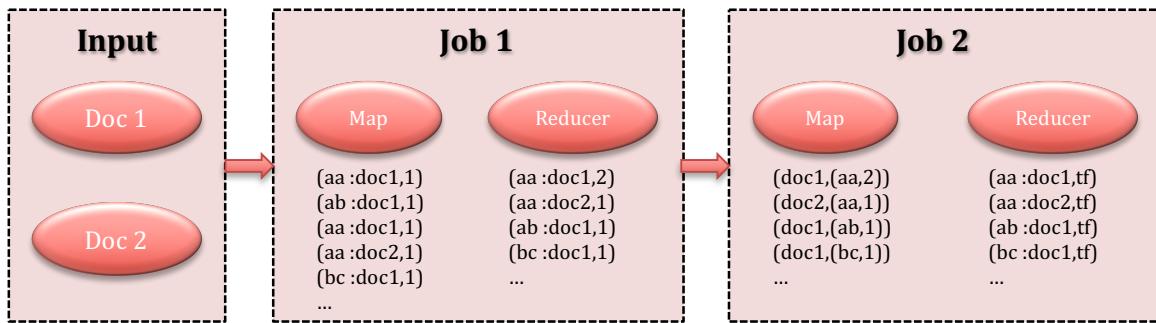
This step aims to calculate : $tf_{wrd} = \frac{wordCount}{wordsPerDoc}$. We define two jobs, let's denote job 1 and job 2.

Job 1

- Mapper result: (word :docName, 1)
- Reducer result: (word :docName, wordCount)

Job 2

- Mapper result: (docName, (word, wordCount)), (produced by remapping the result of the output of Job 1)
- Reducer result: Generates the pair (word :docName, tf)



This step generates an output file of tf as the below screenshot shows (tf :2nd column) :

```

perches:callwild      3.096838128271035E-5
hysteria:callwild    3.096838128271035E-5
attended:callwild    3.096838128271035E-5
yukon:callwild       2.1677866897897247E-4
holding:callwild     6.19367625654207E-5
advances:callwild    1.238735251308414E-4
discharged:callwild  6.19367625654207E-5
answered:callwild    9.290514384813106E-5
pleading:callwild    3.096838128271035E-5
solleks:callwild    3.096838128271035E-5
slope:callwild       9.290514384813106E-5
advanced:callwild   6.19367625654207E-5
...

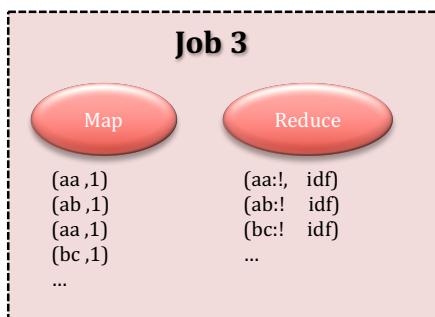
```

• Step 2 (IdfPack.java)

This step aims to calculate : $idf_{wrd} = \log \left(\frac{totalDocs}{DocsPerWord} \right)$. Let's denote job 3 that helps us to generate the result based on the output of the step 1.

Job 3

- Mapper result: (word : 1) (produced by remapping the result of the output of Job 2)
- Reducer result: (word :! idf), which is the idf of the doc 1 or doc 2. (here, ":"! is for easy combination of TF & IDF in IntePack)



This step generates an output file of idf as the below screenshot shows (idf :2nd column) :

```
a: !      0.0
aback: !  0.3010299956639812
abandoned: !   0.3010299956639812
abandoning: !  0.3010299956639812
abandonment: ! 0.3010299956639812
abate: !  0.3010299956639812
abated: !   0.3010299956639812
abatement: ! 0.3010299956639812
abating: !   0.3010299956639812
abed: !   0.3010299956639812
abhor: !  0.3010299956639812
abhorrence: ! 0.3010299956639812
abide: !  0.0
abiding: !   0.3010299956639812
ability: !   0.3010299956639812
abjectly: !  0.3010299956639812
able: !   0.0
...
...
```

• Step 3 (IntePack.java)

This procedure combines the results of step 1 and step 2. Then we could generate an output file of tf-idf as the below screenshot shows (tf-idf :2nd column) :

```
a:callwild    0.0
a:defoe-robinson-103.txt      0.0
aback:callwild  9.322411683254815E-6
abandoned:defoe-robinson-103.txt 9.88279696861396E-6
abandoning:defoe-robinson-103.txt 2.47069924215349E-6
abandonment:callwild  9.322411683254815E-6
abate:defoe-robinson-103.txt 2.9648390905841877E-5
abated:defoe-robinson-103.txt 2.4706992421534895E-5
abatement:defoe-robinson-103.txt 2.47069924215349E-6
abating:defoe-robinson-103.txt 2.47069924215349E-6
abed:defoe-robinson-103.txt 2.47069924215349E-6
abhor:defoe-robinson-103.txt 2.47069924215349E-6
abhorrence:defoe-robinson-103.txt 1.4824195452920939E-5
abide:callwild  0.0
abide:defoe-robinson-103.txt 0.0
abiding:callwild  9.322411683254815E-6
ability:callwild  1.864482336650963E-5
abjectly:callwild  9.322411683254815E-6
able:callwild  0.0
able:defoe-robinson-103.txt 0.0
abode:defoe-robinson-103.txt 7.412097726460469E-6
abominable:defoe-robinson-103.txt 4.94139848430698E-6
abortive:defoe-robinson-103.txt 2.47069924215349E-6
about:callwild  0.0
```

• Step 4 (SortPack.java)

This step sorts the result of td-idf as a descending order.

```
esteemed:defoe-robinson-103.txt 9.88279696861396E-6
seat:defoe-robinson-103.txt 9.88279696861396E-6
profits:defoe-robinson-103.txt 9.88279696861396E-6
bunch:defoe-robinson-103.txt 9.88279696861396E-6
resignation:defoe-robinson-103.txt 9.88279696861396E-6
humble:defoe-robinson-103.txt 9.88279696861396E-6
enterprise:defoe-robinson-103.txt 9.88279696861396E-6
entering:defoe-robinson-103.txt 9.88279696861396E-6
endeavor:defoe-robinson-103.txt 9.88279696861396E-6
enclose:defoe-robinson-103.txt 9.88279696861396E-6
humanity:defoe-robinson-103.txt 9.88279696861396E-6
produced:defoe-robinson-103.txt 9.88279696861396E-6
empty:defoe-robinson-103.txt 9.88279696861396E-6
household:defoe-robinson-103.txt 9.88279696861396E-6
emperor:defoe-robinson-103.txt 9.88279696861396E-6
horn:defoe-robinson-103.txt 9.88279696861396E-6
sedate:defoe-robinson-103.txt 9.88279696861396E-6
embraced:defoe-robinson-103.txt 9.88279696861396E-6
embers:defoe-robinson-103.txt 9.88279696861396E-6
mechanic:defoe-robinson-103.txt 9.88279696861396E-6
...
...
```

Question 5.2 : PageRank

PageRank is an algorithm to measure the importance of website pages.
In our project, we apply the Efficient Computation of PageRank.¹

This program includes :

1) java files

- PageRank_Driver.java : main program
- preparePack.java : adjust the input file data (preprocessing)
- IntePack.java : integrates the “linkpages” file with the new rank results
- contriPack.java : applies the algorithm to calculate the new rank results
- SortPack.java : results sorting after the 30th iteration

2) input file

- sourceinfo

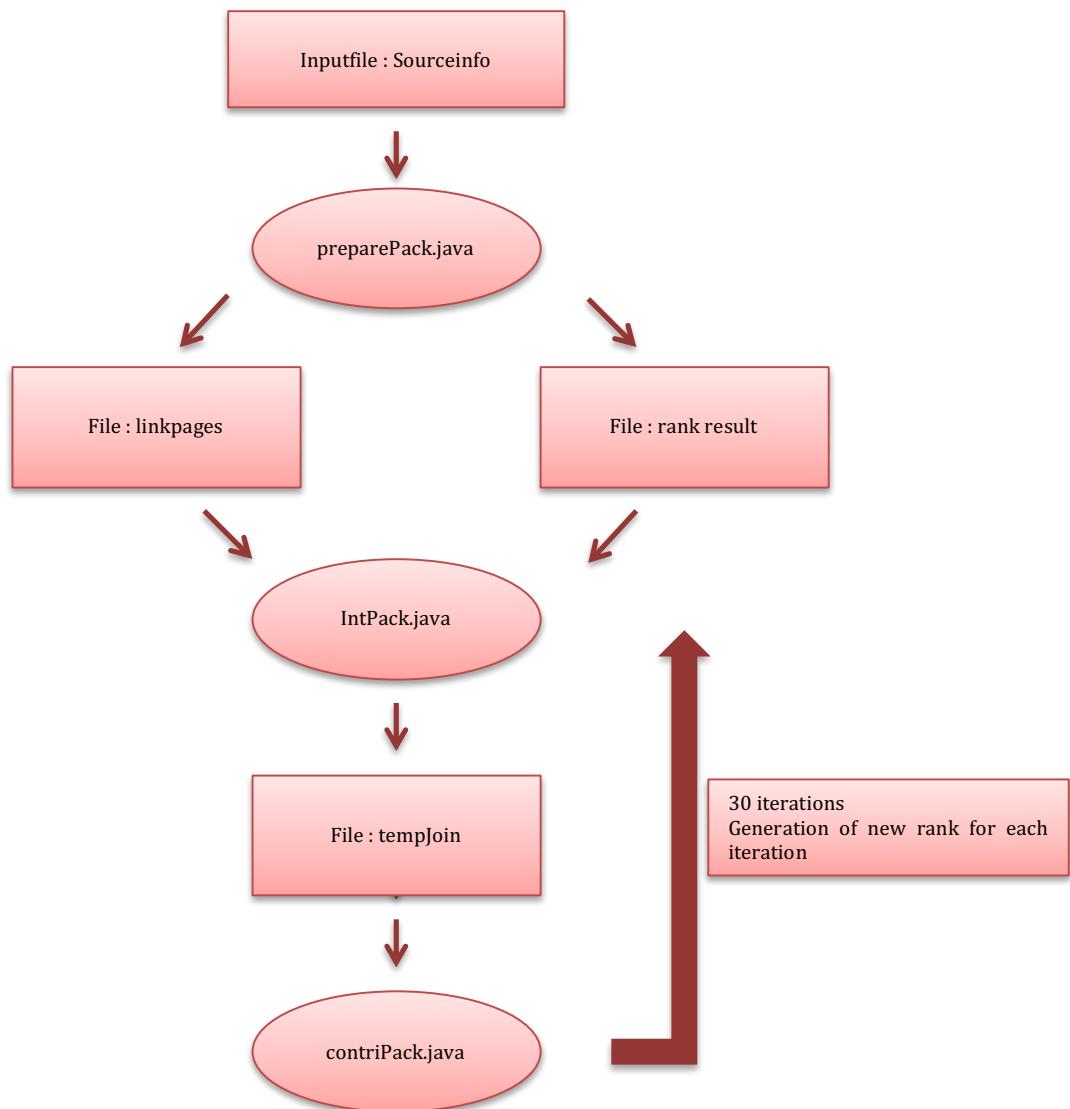
3) output file

- rank i, i=1,...30: the rank result for each iteration
- rans sort : results sorting

4) other files

- linkpages : one result of preprocessing
- rank 0 : the other result of preprocessing, the initial rankings
- tempJoin : the joined file of « linkpages » and « rank » for each time

¹ reference :<http://infolab.stanford.edu/~ullman/mmds/ch5.pdf>, page 178



procedure of question 5.2

- **Step 1 (preparePack.java)**

Firstly, we regroup the data file and get the result named « linkpages », let's take an example as below :

FromNodeID	ToNodeID
0	(1,3)
1	2
2	(1,2)
3	(2,3)

- key =FromNodeID
- value = all the ToNodeID which are connected to this FromNodeID .

We initialize the rankings as below :

NodeID	Rank Result
0	1
1	1
2	1
3	1

- **Step 2 (IntePack.java and contriPack.java)**

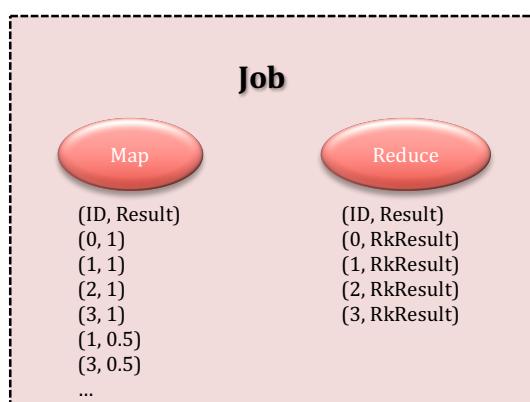
IntePack.java will integrate the data we obtained in the Step 1 and the linkpages, it generates a new « joint data ».

For example, the result of the IntePack for the first time:

FromNodeID	Rank Result
0	(1,3)@1@ ²
1	@1@(2)
2	(1,2)@1@
3	(2,3)@1@

And then calculate the contributions of « ToNodeID » :

contriPack.java will use the new « joint data » of the previous IntePack job, and then generate a new rank result (rank i, with i=1,...30), which is shown in the graph below:



For example, this is the mapper result of the contriPack for the first time:

² we use @ in order to identify the rank result

NodeID	Rank Result
1	0.5
3	0.5
2	1
1	0.5
2	0.5
2	0.5
3	0.5

We run many times IntePack and contriPack and we will get new rank result for each time, which is like this:

NodeID	Rank Result
0	Rk Result ith iteration
1	Rk Result ith iteration
2	Rk Result of ith iteration
3	Rk Result of ith iteration

We obtain the final result of the 30th time :

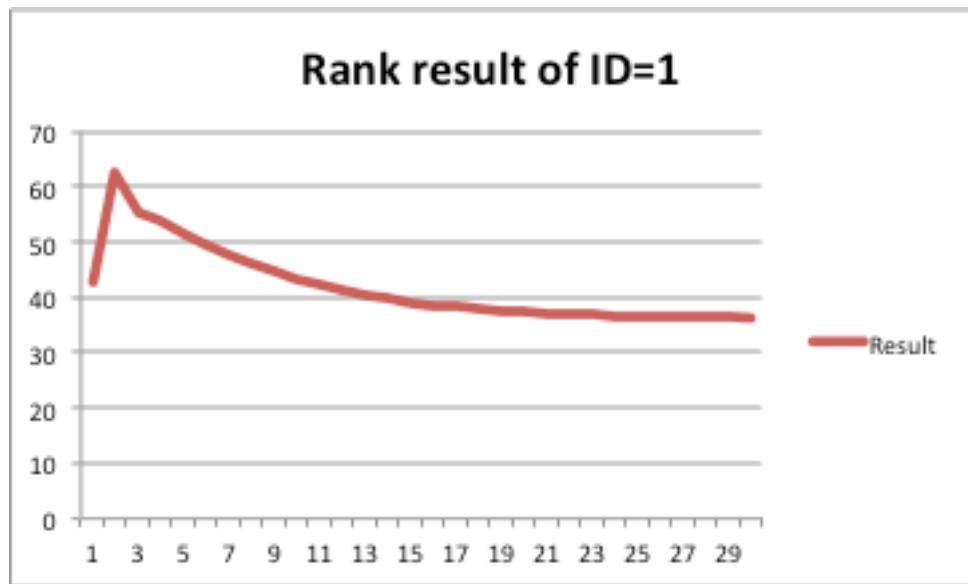
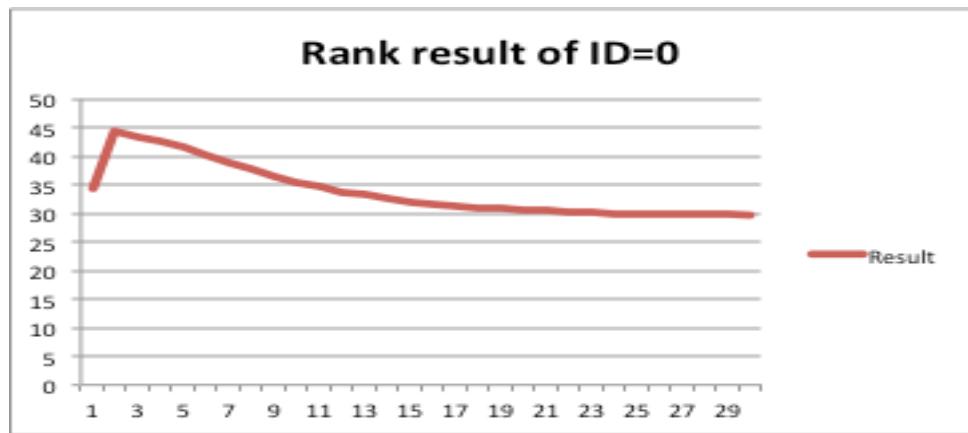
0	@29.679081056999532@
1	@36.26568867518999@
10	@14.253779288194396@
100	@15.679794319785156@
1000	@1.3541512829318476@
10000	@0.4261872467825393@
10001	@0.7639411471753307@
10002	@4.350148557117924@
10003	@2.238645649459073@
1001	@1.9638216058153528@
10010	@0.1963673013153758@
10011	@0.2025987497150164@
10013	@0.18644887218346895@
...	

- **Step 3 (SortPack.java)**

Sorting the result :

737	94.8014137571071
7806	9.939259746319532
852	9.929357426094013
110	9.90909221524188
1763	9.901711754306335
1037	9.891494290548527
2241	9.890586344585413
969	9.87933528870721
2696	9.878017774774001
2110	9.853487637793723
5142	9.84496528350231
1813	9.843118136533938
...	

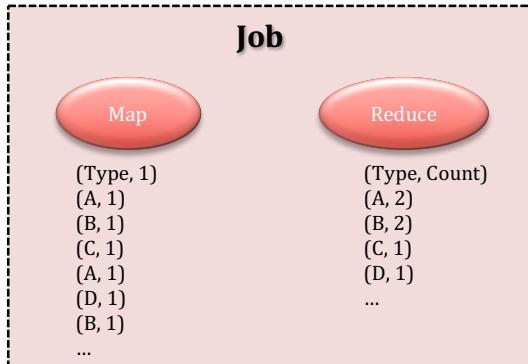
The below graphs show the convergence of the results while the iteration times increase.



Question 5.3 : Trees of Paris

- Compute the number of trees per type

Job

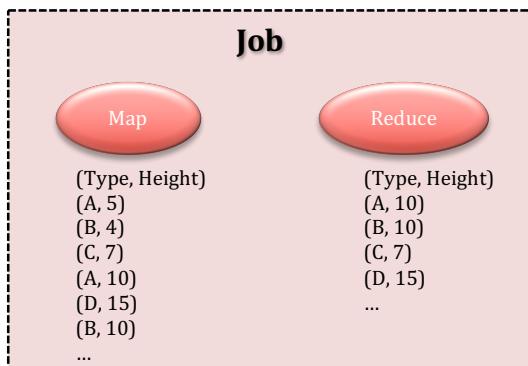


Output

Acer	3
Aesculus	3
Ailanthus	1
Alnus	1
Araucaria	1
Broussonetia	1
Calocedrus	1
Catalpa	1
Cedrus	4
Celtis	1
Corylus	3
Davidia	1
Diospyros	4
Eucommia	1
Fagus	8

- Compute the highest tree of each type

Job

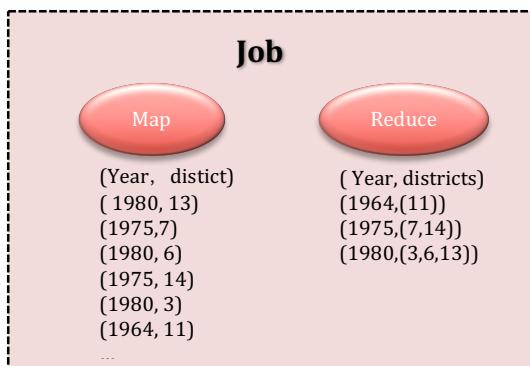


Output

Acer	16.0
Aesculus	30.0
Ailanthus	35.0
Alnus	16.0
Araucaria	9.0
Broussonetia	12.0
Calocedrus	20.0
Catalpa	15.0
Cedrus	30.0
Celtis	16.0
Corylus	20.0
Davidaia	12.0
Diospyros	14.0
Eucommia	12.0
Fagus	30.0
Fraxinus	30.0
Ginkgo	33.0
Gymnocladus	10.0
Juglans	28.0

- Find the district of the oldest tree of Paris (“reduce” will sort the result automatically according to keys)

Job



Output

```

1601      5,
1772      16,
1782      16,
1784      12,
1814      8,7,
1815      12,
1829      12,
1833      20,
1840      17,14,
1843      16,
1845      12,
1847      16,
1850      8,16,16,
1852      16,16,
1854      8,

```