CS 6240
Name: Rushabh Shah
Homework 5
Github Repo: https://github.ccs.neu.edu/rushabh0812/HW5

For each step of the algorithm, briefly explain the main idea for how you partition the problem. Show compact pseudo-code for it. It is up to you to decide how to map the computation steps to MapReduce jobs. For example, you can merge multiple steps into a single job, or you can use multiple jobs to implement a single step. Make sure you discuss and show pseudo-code for both versions of step 3.
Ans:

Step 1:
Create matrix $\mathbf{M}'$ from either the original input or the adjacencylist representation from a previous assignment. (But also see the discussion about sparse matrix representation below!) We create the adjacency lists as we have done before in previous assignments. i.e. PageName: Adjacency List.
The reduce call performs following functions
a) Compute dead links
b) Offset w.r.t to each worker
c) Getting local pageName: Number map.
The partitioning ensures that each worker gets 1/nth of dataset
The pseudo code for Step 1 can be visualized as follows :

```
Map (key k, line l)
{
    nodeId = getKey(l) // Parser Given as part of homework
    adjList = getAdjacencyList(l)
    emit(nodeId, adjList)
}


Class Reduce{
    Method Setup(){
    Counter = 0; HashMap nameToNumMap;
    }

    method reduce(node_name n, List[adjLists] )
    {
        StringToNumMap.put(n, Counter++)
```

```
                for each adjList in adjLists
                aggrAdjLinks += adjList
                emit(n, aggrAdjLinks)
         }

        method Cleanup(){
                emit(MapId, StringToNumMap emit(MapId, Counter)
        }
} //end of reduce class
```

Now, using the offset and local map, we create a unique global map between pageName and Number. This is easily done using a map only job. As before , the job is partitioned such that each worker gets to compute global map for 1/nth of inputs. However every map sees complete offset data using distributed cache. The pseudo code for the above process is as follows:

```
Class Mapper {
HashMap globalMap ;
HashMap offset

Method Setup(){
HashMap offset = getOffset(pathToHdfs);
}

Method map(id, localMap){
for each element e in localMap:
        globalMap.put(e.name,e.number+offset(id))
}

Method cleanup(){
Emit(globalMap);
Emit(inverted(globalMap));
 }
 }
```

After this step, we have all the files needed to construct sparse matrix. We do this in two ways

Method 1: Column Partition:

To form a column we just need adjacency list and hence it is a map only job. As before, we partition the data so that each worker gets 1/nth of data. The pseudo code is as follows:

```
Class Map{
Method setup(){
Map nameToNumberMap = load Map from hdfs
 }
method map(String pageName, List[Adj1, Adj2 .. ]){
 for( each adj in list){
List rowContribution += [(nameToNumberMap (adj)]
}
emit(nameToNumberMap (pageName), rowContribution); if(list.size == 0){
emit(nameToNumberMap.get(pageName))
}
}
```

Method 2 : Row Partition

```
Class Map{
setup(){
Map NameToNumber = load from hdfs
}
method map(String pageName, List[Adj1, Adj2 .. ]){
for( each adj in list){
emit(NameToNumber.get(adj), (NameToNumber(pageName), 1/list.size))
 }
if(list.size == 0){ // write to dangling vector on HDFS
emit(NameToNumber.get(pageName))
}
 } cleanup(){ // Do Nothing } }


method Reduce(PageNumber, [(PageNumber, Contributions),List]){
Emit(PageNumber, List)
}
```

Step 2: Create vector **R**(0) with the initial PageRank values 1/|*V*|. Created automatically during 0 th iteration.

Step 3: Run 10 iterations to compute **R**(10).

Method 1 : Column Partition

We use two jobs to perform the actions

Job1 :

```
Method Map(Number, Contribution){
emit(ColNumber, RowContributions)
emit(PageNumber, PageRank)
emit(DanglingColNumber, null)
}

class Reducer{
HashMap aggregator;
LocalDanglingMass;

Method setup(){ //do nothing }

method reduce(Number, [Contributions…]){
double pageRank = getPageRankEntryFrom[Contributions];
for (each Contribution in Contributions){ if(ValidContribution){
aggregator[Contribution.row] += Contribution.value*pageRank;
 }
}

CleanUp(){ Emit(localDanglingMass); Emit(aggregator); }
```

Job 2:

```
Method Map(Number, Contribution){
Emit(RowNumber, Contributions)
}

method Combiner(RowNumber,List [Contributions]){
Sum = 0
For each Contribution c in List {
Sum = sum + c
}
emit(RowNumber, Sum)
}
```

```
class Reducer{
method Setup(){
GlobalDanglingMass = input from output of job 1
}

reduce(Number, List [Contributions…]){
Sum = 0
For each Contribution c in List :
Sum + = Contribution
PageRank = 0.15/NumberOfPages + 0.85 *(sum + GlobalDanglingMass)
Emit(Number, PageRank)
}
```

Method 2: Row Partition. This method is a map only job

```
Class Map{
Map PageRanks
Double danglingMass = computeDanglingMass

setup(){ //do nothing }

map(Number pageNum, List[(num, contribution) .. ]){
contributionAggr = 0
for( each (num, contribution) in list){
contributionAggr += PageRanks.get(num) *
}
PageRank = 0.15/n + 0.85 (contributionAggr + danglingMass )
emit(PageNameToNumber(pageName), rowContribution);
}
cleanup(){ //do nothing }
}
```

Step 4: Return the top-100 pages from **R**(10).

This step is implemented using Top-K design pattern and uses same approach mentioned in modules.
The pseudo code from modules is as follows:

```
Class Mapper{ localTopK
```

```
setup(){ initialize localTopK
}
map(... , x){ if (x is in localTopK)
// Adding x also evicts the now (k+1)-st record from localTopK
localTopK.add(x) }
cleanup(){ for each x in localTopK
emit(dummy, x) }
reduce(dummy, [x1, x2, x3....]){ initialize globalTopK
for each record x in input list if (x in gloablTopK) // Adding x also evicts the now (k+1)-st record from globalTopK globalTopK.add(x)
for each x in localTopK emit(NULL, x)
}
```

Briefly explain how each matrix and vector is stored (serialized) for versions A and B. Do not just paste in source code. Explain it in plain English, using a carefully chosen example. See the above discussion about dense and sparse matrices for an example how to do this.

Ans:

The serialization for column partition is as follows

ColumnNumber1 [(RowNumberI,Contribution)…….]

ColumnNumber2 [(RowNumberJ,Contribution)…….] . . .
ColumnNumberN[(RowNumberA,Contribution)…….]

Example 10[(6,0.4),(4,0.1),(7,0.7)]

The serialization for row partition is as follows

RowNumber1 [(ColumnNumberI,Contribution)…….]

RowNumber2 [(ColumnNumberJ,Contribution)…….] . . .

RowNumberN[(ColumnNumberA,Contribution)…….]

Example 10[(6,0.4),(4,0.1),(7,0.7)]

Discuss how you handle dangling nodes: Do you simply replace the zero-columns in M and then work with the corresponding M′, or do you deal with the dangling nodes separately? Make sure you mention if your solution for dangling nodes introduces an additional MapReduce job during each iteration.

Ans: The dangling mass contribution for each row is equal to D*R which is a constant. Hence we calculate it once per iteration.

Report for both configurations the total execution time, as well as the times for (i) completion of steps 1 and 2, (ii) time for an iteration of step 3, and (iii) time for step 4. Make sure you clarify if your program works with the original input files or with the parsed adjacency lists from a previous assignment. Since there are two versions (A and B) for step 3, you need to report 4 sets of running-time numbers.

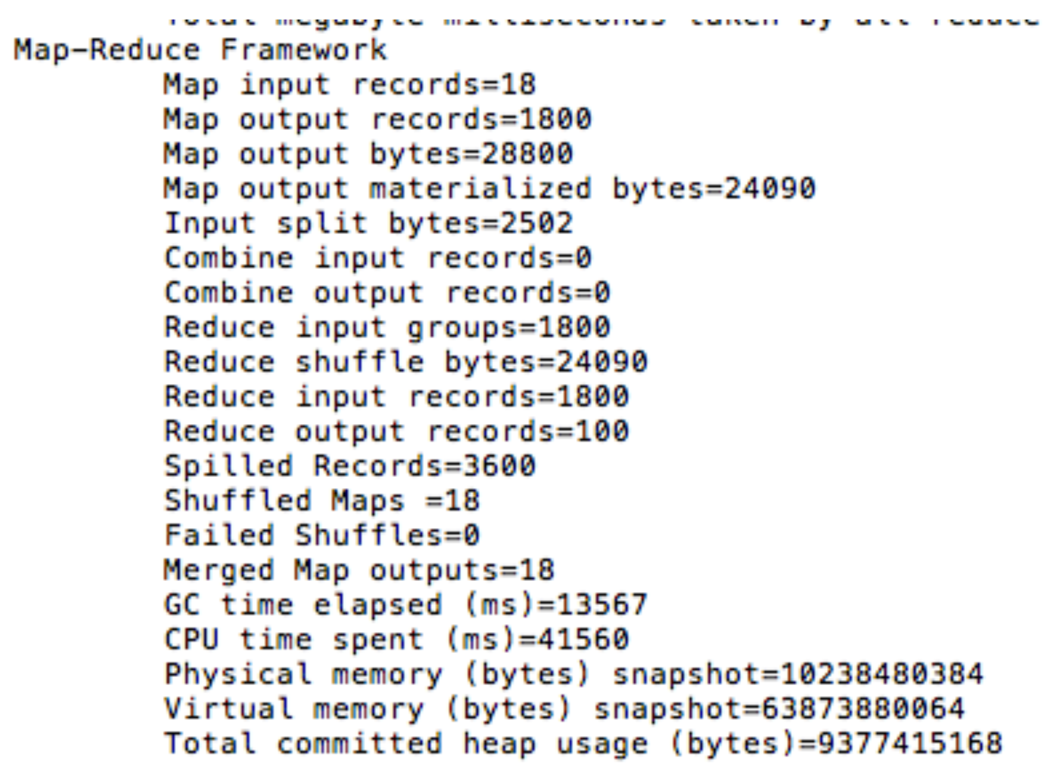
Ans: The solution works with original input files.

	Column		Row		Adjacency	
	6 machines	11 machines	6 machines	11 machines	6 machines	11 machines
Step 1 & 2	1861285 ms	1073273 ms	1925322 ms	997864ms	---------	--------
Step 3	223412ms	151263ms	56874ms	39062ms	129873ms	94558ms
Step 4	46128ms	45826ms	42564ms	43756ms	---------	-------
Total	3924162 ms	2602148 ms	2506681 ms	1611825 ms	----------	---------

For each configuration, report the running time of an iteration executed by the adjacency-list based Hadoop implementation from the earlier HW assignment. How do these numbers compare to the adjacency-matrix based version? Briefly discuss if you find the result surprising and explain possible reasons for it. Make sure you back your arguments with concrete data, e.g., from the log files.

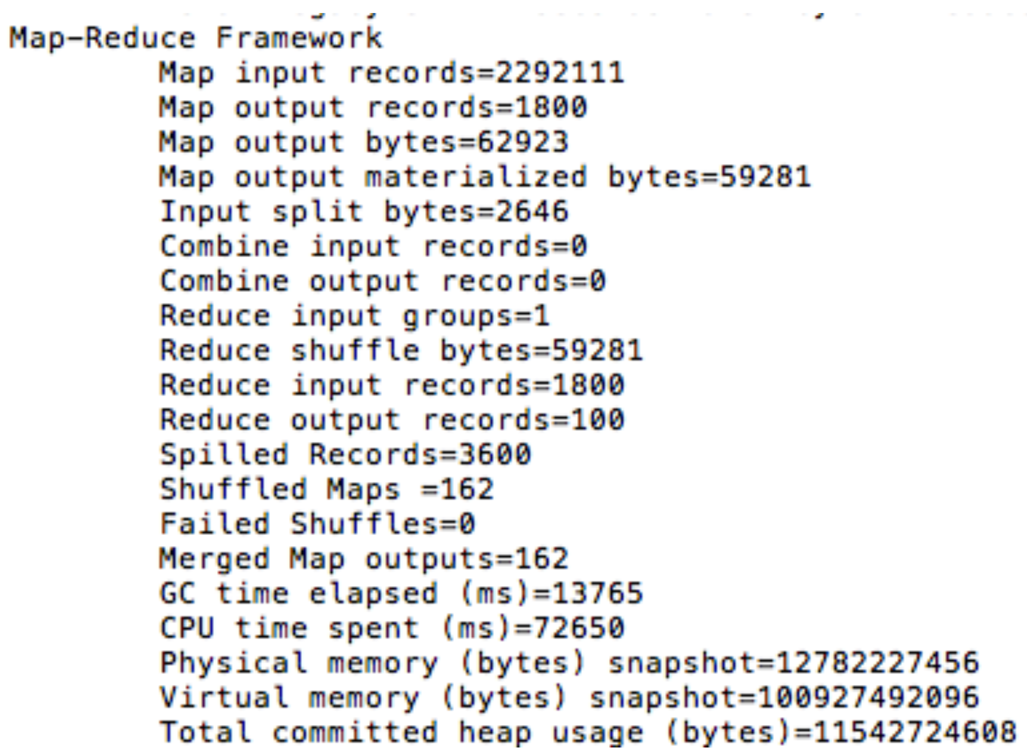
The results are in-line with our expectation. In general, row based partition performs fastest whereas adjacency-list based implementation is second . The column-based approach is the worst performing.

Row Partition:



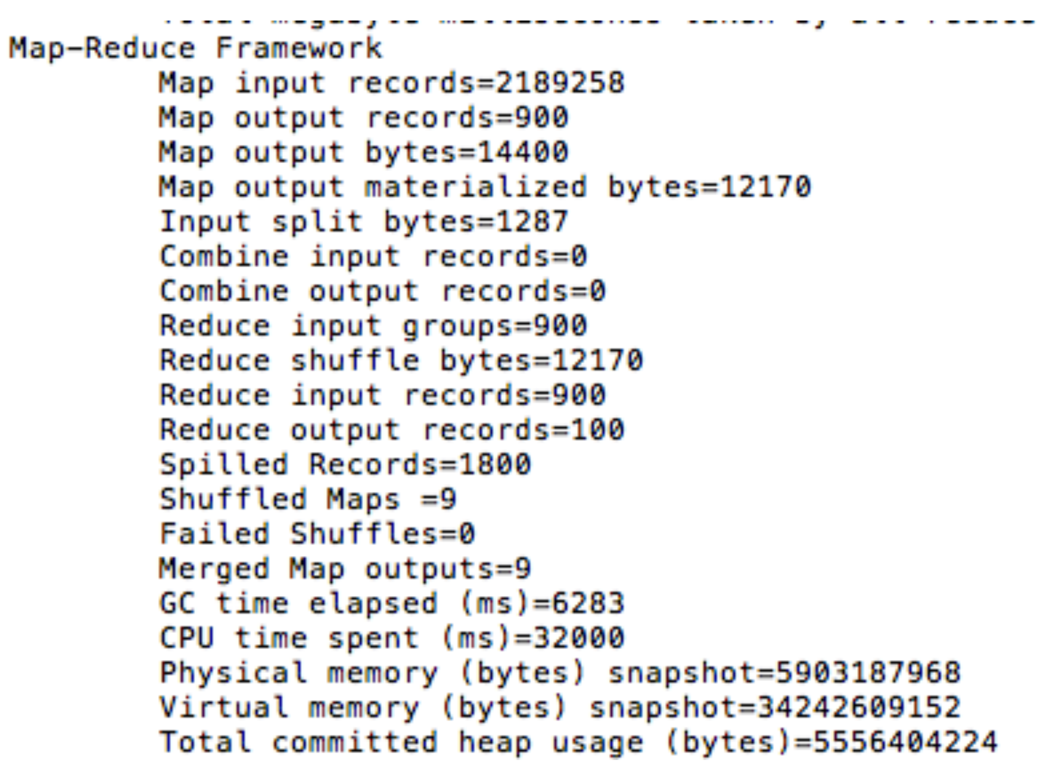
```
 Map-Reduce Framework
         Map input records=18
         Map output records=1800
         Map output bytes=28800
         Map output materialized bytes=24090
         Input split bytes=2502
         Combine input records=0
         Combine output records=0
         Reduce input groups=1800
         Reduce shuffle bytes=24090
         Reduce input records=1800
         Reduce output records=100
         Spilled Records=3600
         Shuffled Maps =18
         Failed Shuffles=0
         Merged Map outputs=18
         GC time elapsed (ms)=13567
         CPU time spent (ms)=41560
         Physical memory (bytes) snapshot=10238480384
         Virtual memory (bytes) snapshot=63873880064
         Total committed heap usage (bytes)=9377415168
```

Adjacency List:



```
Map-Reduce Framework
        Map input records=2292111
        Map output records=1800
        Map output bytes=62923
        Map output materialized bytes=59281
        Input split bytes=2646
        Combine input records=0
        Combine output records=0
        Reduce input groups=1
        Reduce shuffle bytes=59281
        Reduce input records=1800
        Reduce output records=100
        Spilled Records=3600
        Shuffled Maps =162
        Failed Shuffles=0
        Merged Map outputs=162
        GC time elapsed (ms)=13765
        CPU time spent (ms)=72650
        Physical memory (bytes) snapshot=12782227456
        Virtual memory (bytes) snapshot=100927492096
        Total committed heap usage (bytes)=11542724608
```

Column Partition:



```
Map-Reduce Framework
        Map input records=2189258
        Map output records=900
        Map output bytes=14400
        Map output materialized bytes=12170
        Input split bytes=1287
        Combine input records=0
        Combine output records=0
        Reduce input groups=900
        Reduce shuffle bytes=12170
        Reduce input records=900
        Reduce output records=100
        Spilled Records=1800
        Shuffled Maps =9
        Failed Shuffles=0
        Merged Map outputs=9
        GC time elapsed (ms)=6283
        CPU time spent (ms)=32000
        Physical memory (bytes) snapshot=5903187968
        Virtual memory (bytes) snapshot=34242609152
        Total committed heap usage (bytes)=5556404224
```

Top 100 Page Ranks:

Full Dataset:

0.002895612340091925 United_States_09d4
0.002583766265837734 2006

```
0.0013756641642095888 United_Kingdom_5ad7
0.0011887559248920127 2005
9.331545921969787E-4 Biography
9.001438659434939E-4 Canada
8.946269909617604E-4 England
8.819707744034018E-4 France
8.274616141237871E-4 2004
7.562886831189644E-4 Germany
7.364585608851388E-4 Australia
7.160204348146395E-4 Geographic_coordinate_system
6.664083680789282E-4 2003
6.491299090528711E-4 India
6.337472702443232E-4 Japan
5.378696331109533E-4 Italy
5.36148702704579E-4 2001
5.288554115331097E-4 2002
5.26108855277083E-4 Internet_Movie_Database_7ea7
5.053386276543934E-4 Europe
5.015343994390617E-4 2000
4.815731814636354E-4 World_War_II_d045
4.677556736351052E-4 London
4.4795412462642184E-4 Population_density
4.458486273871629E-4 Record_label
4.4299247468122354E-4 1999
4.3990978261100285E-4 Spain
4.395693315378872E-4 English_language
4.146181651548827E-4 Race_(United_States_Census)_a07d
4.13649536881574E-4 Russia
4.0666077968928675E-4 Wiktionary
3.8530050980005014E-4 Wikimedia_Commons_7b57
3.827812361069544E-4 1998
3.7512127662490095E-4 Music_genre
3.656526984096312E-4 1997
3.610261602059302E-4 Scotland
3.604573587598679E-4 New_York_City_1428
3.438433136333573E-4 Football_(soccer)
3.431247424875291E-4 1996
3.3844560924637486E-4 Television
3.3800039350956475E-4 Sweden
3.269634115654665E-4 Square_mile
3.2627172674296387E-4 Census
3.229966918582883E-4 1995
```

```
3.2165967950409553E-4 California
3.164946807584972E-4 China
3.1133725323645456E-4 Netherlands
3.1072367472309776E-4 New_Zealand_2311
3.084883710906191E-4 1994
2.9367186085489124E-4 1991
2.9131985809068125E-4 1993
2.8938672813236846E-4 1990
2.8901262991029103E-4 New_York_3da4
2.884412303901588E-4 Public_domain
2.7906163398987936E-4 1992
2.7877882733432744E-4 United_States_Census_Bureau_2c85
2.778208315536229E-4 Film
2.759587905167049E-4 Scientific_classification
2.7540669173471856E-4 Actor
2.7257320340226147E-4 Norway
2.716799576146704E-4 Ireland
2.649319762740503E-4 Population
2.6177712979952956E-4 1989
2.557897929504716E-4 1980
2.5559028895623127E-4 Marriage
2.550329430472775E-4 January_1
2.542917856354147E-4 Brazil
2.5294642598586706E-4 Mexico
2.5191426125748754E-4 Latin
2.490248595067059E-4 1986
2.4699658689432E-4 Politician
2.4277841622043016E-4 1985
2.4240004633174913E-4 1979
2.418948989627153 1E-4 1982
2.4173203771540535E-4 French_language
2.4157035425746514E-4 1981
2.4117545225029894E-4 Per_capita_income
2.3980906233595788E-4 Poland
2.3948752284683503E-4 1974
2.393706993389868E-4 Album
2.3779856609789502E-4 South_Africa_1287
2.372284536609572E-4 Switzerland
2.3714836890721132E-4 1984
2.371475854077191E-4 1987
2.3698196286943223E-4 1983
2.3576252382891895E-4 Record_producer
```

```
2.3314836602587803E-4 1970
2.3176630161779772E-4 1988
2.3040742074454954E-4 1976
2.291697977116413E-4 Km²
2.274990546775559E-4 1975
2.2478568148505255E-4 1969
2.2454623105836683E-4 Paris
2.2348383281810453E-4 Greece
2.232404651898998E-4 1972
2.224573575150376E-4 Personal_name
2.2197817108490686E-4 1945
2.2135127507517773E-4 1977
2.2040350997510724E-4 Poverty_line
2.2034742559309798E-4 1978
```

Simple Dataset:

```
0.006270017474887574 United_States_09d4
0.004746521647459629 Wikimedia_Commons_7b57
0.003879860727134216 Country
0.0026809443776084126 England
0.0026073381113115393 United_Kingdom_5ad7
0.0025985862902878584 Europe
0.002583908182959156 Water
0.0025349693900029984 Germany
0.0025115882823210393 France
0.002457673194457764 Animal
0.002423856703657672 Earth
0.0023498665911180962 City
0.0020076003706547673 Week
0.0019192207972314262 Asia
0.0018681364480375954 Sunday
0.001861901316056162 Wiktionary
0.0018410203611934376 Monday
0.0018359686351703906 Money
0.001822808684663094 Wednesday
0.001814186946073507 Plant
0.0017783378690091481 Friday
0.0017633056713257354 Computer
0.0017586117009649052 Saturday
0.0017474216199944413 English_language
0.0017359752571455788 Thursday
```

```
0.0017235388050616002 Tuesday
0.0017146957999186524 Italy
0.0017018287899640502 Government
0.0017010255471243644 India
0.0015895381781201264 Number
0.0015576674302577693 Spain
0.0015154558546106004 Japan
0.0014979884203457005 Canada
0.0014706001107316843 Day
0.0014454183366565591 People
0.0014190431134654756 Human
0.0013736995548105902 Wikimedia_Foundation_83d9
0.0013667870686872427 Australia
0.0013655763518341383 China
0.0013354508748833038 Energy
0.0013193432122939341 Food
0.0012950569993510342 Sun
0.0012918919151513753 Science
0.0012775185841551403 Mathematics
0.0012282471356595299 index
0.0012273536593279241 Television
0.0011882683246315603 Capital_(city)
0.001180800001039206 Russia
0.001163827941607845 State
0.0011570661576948671 Music
0.0011358675592467794 Year
0.0011122234112611743 Greece
0.0011067015768364543 Scotland
0.0011041364469016738 Language
0.0010840168781008702 Metal
0.0010701037513338616 Wikipedia
0.001062271736905719 Greek_language
0.0010512801859716356 2004
0.0010320329748448635 Planet
0.0010256749088513196 Sound
0.0010237591961046399 Religion
0.0010208486668707754 London
9.919686691150947E-4 Africa
9.544869629369787E-4 20th_century
9.500184614578623E-4 Law
9.430356425999924E-4 Geography
9.388921587115984E-4 Liquid
```

```
9.365804297985694E-4 19th_century
9.251877891523699E-4 World
9.143875732073565E-4 Scientist
9.108073281798569E-4 Society
8.797103674045316E-4 Atom
8.789441517334165E-4 Latin
8.75635201122971E-4 History
8.685252408615899E-4 War
8.684172930340793E-4 Sweden
8.68121350969594E-4 Poland
8.662307127425791E-4 Light
8.582735823099305E-4 Netherlands
8.48816292445545E-4 Culture
8.411601572134674E-4 Building
8.24762979570608E-4 God
8.19208677747904E-4 Turkey
8.169803862799111E-4 Plural
8.125725939261813E-4 Information
8.057041813788026E-4 Centuries
7.947276814964444E-4 Chemical_element
7.906159051543348E-4 Portugal
7.809073542051401E-4 Inhabitant
7.771310938113499E-4 Denmark
7.735863657245808E-4 Capital_city
7.703195785370825E-4 Austria
7.586412399089746E-4 Species
7.578020413576398E-4 Cyprus
7.566499018373539E-4 Ocean
7.550868120552983E-4 Book
7.550238738104447E-4 Disease
7.536968663668928E-4 North_America_e7c4
7.495640356183104E-4 Biology
7.495514164825975E-4 University
```